MOTION PLANNING FOR CONSTRAINED MOBILE ROBOTS IN UNKNOWN ENVIRONMENTS

LAI XUECHENG
(B.Eng, Zhejiang University)
(M.Eng, Zhejiang University)

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING
NATIONAL UNIVERSITY OF SINGAPORE
2007

Acknowledgements

I am grateful to all the people who have encouraged and supported me during my PhD study, which led to this thesis. Firstly, I am full of gratitude to my supervisor, Professor Shuzhi Sam Ge, for his constant and patient guidance, and inspiration, especially for his selflessly sharing his invaluable experiences and philosophies in and beyond research. I sincerely thank my second supervisor, Assistant Professor Abdullah Al Mamun, for his constant guidance, help, and support during my PhD study. Without their guidance or support, I would not have made this thesis accomplished.

At work I have had the great fortune of working with brilliant people who are generous with their time as well as being good friends. Special thanks must be made to Mr Fua Cheng Heng and Mr Tee Keng Peng, with whom a number of discussions on research have been made. Lots of thanks to Dr Wang Zhuping for her friendly help and willing guidance ever since the day I joined NUS. Thanks to Mr Kong Cheong Yeen and Mr Ong Phian Ting, who worked closely with me and contributed much valuable programming and experiment work during their FYP projects. Thanks Mr Jiang Jinhua and other VIP students for all your help with setup of Linux and our Magellan Pro robot.

Thanks to Mr Tian Zhiling, Ms Liu Jing, Dr Chao Zhijun, Dr Tang Kok Zuea, Dr Xiao Peng, for providing valuable help for my research to progress. Special thanks to Assistant Professor Nicholas Roy, who has continuously provided answers or clues to my questions related to CARMEN based development. Thanks Dr Jia Li for your friendship. Thanks to Mr Yang Yong, Mr Wang Liwang, Mr. Wu Zhenyu, Mr Tao Pey Yuen, Mr Han Thanh Trung, Mr Yang Chenguang, Mr Wang Huafeng, Mr Guan Feng, Mr Zhao Guang, Dr Chen Xiangdong, Dr Huang Loulin, Dr Zhang Jin, and other fellow students/colleagues for their help and friendship.

I am thankful to the National University of Singapore for providing me with the research scholarship to undertake the PhD study.

Last but not least, I would like to thank my wife, my parents and my sister for their generous and unconditioned support. Their example has always been the source of my ambitions.

Abstract

This thesis considers developing a framework of and the associated technical issues with path planning and motion planning in unknown or partially known environments for mobile robots subject to various constraints.

The research addresses how to navigate a robot to the destination without *a priori* information about obstacles. Both purely-sensor-based and map-aided approaches, combined with different strategies, are considered in order to accomplish global convergence to the goal, the primary aim of a path planning task. One effort is to develop a technique for guiding a physical robot to continuously follow an obstacle of arbitrary shape in the desired direction, knowing that a proper combination of it with moving straight forward to the goal may eventually lead the robot to its destination. The other one is to gradually build a model of the environment and search for a possible route to the goal with the periodically updated model, while a suitable technique is explored in order to drive the robot to the waypoints (which the route consists of).

Robot dynamic constraints and requirement of a smooth motion pose additional challenges to the above research – a physical robot might not be able to track the velocities commanded, which may introduce problems not only to the motion planning itself but also to the safety of the robot or the surroundings. In view of this, these constraints are taken into account in the process of path planning (where the major concern is to find a smooth path satisfying the various robot constraints) or motion planning (where the major concern is to generate an optimized, waypoint-directed motion command satisfying a certain objective function).

With the above objectives and guidelines in mind, the technical contributions of this thesis are generally applicable to a wide variety of sensor-based path/motion planning, online map building, simultaneous mapping building and path planning, and motion planning considering robot dynamics problems. The proposed solutions, which are demonstrated theoretically and empirically, include:

i) A practical approach of boundary following for a mobile robot with limited sensing ability. The robot follows an obstacle of arbitrary shape continuously in the desired direction by locating a series of Instant Goals. Based on it, a practical globally convergent path planner is presented for mobile robot navigation in unstructured, complex environments.

ii) A polar polynomial curve method for smooth, feasible path generation for non-holonomic mobile robots with collision test carried out in real-time. The path and associated velocity profile are generated such that dynamic and curvature constraints are satisfied. Based on it, a sensor-based hybrid approach is proposed for smooth path planning for differential drive robots.

iii) A simple yet efficient methodology of automatic online map building for mobile robots. Laser and sonar data are fused in a selective way to produce a better representation of the environment and to achieve better obstacle detection.

iv) A hierarchical framework for incremental path finding and optimized dynamic motion planning in unknown environments. It searches a periodically updated map containing unknown information and finds a global optimal path robustly. To trace the path, an optimized motion minimizing a situation-dependent objective function is searched within a one-dimensional velocity space such that the robot can move at a relatively high speed and effectively avoid collision with obstacles.

Nomenclature

$\overrightarrow{AB}$	straight-line segment starting from point A to point B;
$\overline{AB}$	non-directional straight-line segment with end points A and B;
$\lvert AB \rvert$	straight-line distance between point A and point B;
$\overrightarrow{A(B)}$	straight-line starting from point A and passing point B;
$\widehat{AB}$	circular arc starting from point A to point B;
$\mathbf{n}_{AB}$	unit vector pointing from A to B;
$\Upsilon_A^B(r)$	rectangular ray expanded from line segment $\overrightarrow{AB}$ by a radius r;
$\mathcal{C}(A, r)$	circular disk centered at point A and with radius r;
(x, y)	coordinates with respect to (w.r.t.) the global frame;
ϑ	angle w.r.t. the global frame, or orientation (angle of the main axis) of the robot;
$\mathbf{q}$	configuration of the robot;
(x_b, y_b)	coordinates w.r.t. the body frame attached to the robot;
θ	angle;
RP	reference point, the fixed point designated on a robot for it to track a path/route;
v	linear velocity/translation velocity (defined at the reference point for a robot to trace a path);
ω	angular velocity/rotation velocity;
$\mathcal{V}$	front-wheel velocity, which is defined as the velocity of the mid point of the front axle of a car-like robot;
φ	steering angle of the front wheels of a car-like robot w.r.t. the body frame;
a_l and a_n	longitudinal and centripetal accelerations of the robot;
a	longitudinal acceleration of the robot;
ε	angular acceleration of the robot;
IC	instant center, turning center of the robot;
IG	Instant Goal;
Θ	search range to obtain an Instant Goal;
r	turning radius (distance from the turning center) of the robot;
κ	signed curvature of a curve;
R_j	measured distance of the obstacle detected in the jth direction, where $j = 1, 2, \ldots, N_s$ (N_s is the number of laser readings in one scan);
ρ	distance from the robot coordinate frame;
Ω	origin of a polar coordinate frame;

ϱ and ϕ	polar radial and polar angle in a polar coordinate frame (origin not at the robot), respectively;
Φ	polar angle span of a polar polynomial curve;
S, G	initial position of the robot, and the goal;
P	point or its coordinates;
μ	friction coefficient between the wheel tires and the ground;
Δt	sampling interval, or period between two successive moving actions;
t	current time instant;
t_0, and t_f	initial and final time instant for the robot to travel between two configurations;
$s(t)$	arc length of robot trajectory from time 0 to time t;
τ	duration of the motion for the robot to travel between two configurations, i.e. $\tau = t_f - t_0$;
P_t and ϑ_t	position and orientation of the robot at the time instant t.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter presents motivation and background for carrying out the research work of this thesis, which is on path planning, motion planning and map building for autonomous mobile robots, followed by the research objectives and scope of this research as well as the outline of this thesis.

1.1 Motivation of Research

In recent years, research on autonomous mobile robots has been one of the key focuses in the robotics and automation community. Autonomous mobile robots have a number of military, industrial, and domestic applications. They can be used in hazardous environments to save manpower cost and accomplish tasks dangerous to human beings, such as nuclear plant maintenance or search and rescue after disaster. In order for a mobile robot to accomplish a designated mission independently, one of the fundamental functions that it must have is path planning, which is to obtain a feasible and safe path for the robot to navigate from its initial position to its destination without collision with the obstacles in the environment [1]. Path planning and motion planning can either be treated separately as two sequential tasks, or be dealt with as a single task, i.e., motion planning is to generate appropriate control commands for the robot to move toward the destination and avoid obstacles simultaneously.

Many achievements have been gained in the areas of robotics research, especially in the fields of positioning and map building, and path planning during the past two decades. Yet, much more work is needed to develop new theories and algorithms because of the following complexities in achieving truly autonomous robot navigation:

- **Complexity in mobile robots**: mobile robots are typically subject to various robot constraints (kinematics, dynamics, etc.) and uncertainties in control; and
- **Complexity in environmental modeling**: no *a priori* or only limited knowledge of the environment is available, and uncertainties/errors exist in sensing and positioning inevitably.

The subsequent sections will present background and literature review on motion/path planning for mobile robots, focusing on achieving global convergence[1] of path planning and considering robots constraints in motion planning.

1.2 Global Convergence of Path Planning

Depending on whether there is complete information about the environment that the robot navigates in, path planning of mobile robots can be divided into two categories [2]: path planning with complete *a priori* knowledge of the environment, and path planning with no *a priori* or partial knowledge of the environment. Both categories of approaches have their own pros and cons.

1.2.1 Sensor-based Path Planning

When no complete *a priori* knowledge of the environment is available, on-board sensors have to be employed to perceive the environment, in order for a robotic system to accomplish online navigation or path planning. A number of local approaches have been proposed which do not build a global world model. Potential field methods [1,3–5] initiated by [6] have been extensively studied because of its efficiency and mathematical elegance. Classical potential field methods involve an artificial force acting upon the robot, derived from the vector summation of an attractive force representing the goal and a number of repulsive forces associated with the individual known obstacles. A successful local approach called vector field histogram (VFH) method [7] uses the histogram built from obstacles to achieve fast obstacle avoidance for mobile robots. Other local approaches include subsumption method [8], "behavior-based approaches" [9,10], and the "wander" routine (which generates random movements for the robot). Most of them adopt a strategy of behavior-based

[1] If the goal is reachable, the path planner generates a continuous collision-free path from the start location to the goal; otherwise, it reports the failure.

reaction to the local environment information for robot navigation.

Local approaches handle uncertainty and unpredictable changes well by giving up the idea of modeling and reasoning about the environment and the future consequences of actions. Some of them have been used successfully in obstacle avoidance. However, the robotic systems based on such approaches have the following drawbacks: (i) behavior-based systems are memoryless, which leads to the phenomenon of repeated behavior or cycles without optimization of the path [5]; and (ii) the robot may be stuck in a local minimum without being able to get out of it. The likely occurrence of cyclic behaviors as well as local minima make any system that relies solely on local navigation approaches somewhat unreliable.

Harmonic potential functions [11,12] were used to solve the local minima problem of potential field methods, based on the assumption of full *a priori* knowledge of the goal's location and of the obstacles' trajectories. Randomized path planning methods [13,14] have been successfully applied with the central concept being that the robot's free space is not explicitly represented but randomly sampled. Hsu *et al.* [15] presented a randomized motion planner for robots to achieve a specified goal under kinematic and dynamic constraints while avoiding collisions with moving obstacles with known trajectories. The probability that such methods find a path when one exists increases with their running time. However, global convergence is not guaranteed. Moreover, in order to build a *roadmap*, complete or partial knowledge of the environment must be provided beforehand.

In order to achieve global convergence, Lumelsky and Stepanov [2] proposed Bug algorithms which keep the robot switching between two motion modes: moving directly to the goal, and following an obstacle. The algorithms require only limited global information (e.g. the robot position), in addition to local sensory information. Later, Lumelsky and Skewis [16] incorporated range sensing into their robot navigation function based on the Bug algorithms. Some variations of the Bug algorithms have been proposed, such as DisBug algorithm [17], TangentBug [18], 3DBug algorithm [19] and "pursuit-evasion" Bug algorithm [20]. In these algorithms, transition between the two motion modes is governed by a criterion ensuring the distance to the goal decrease monotonically. In this way, the problem of path planning with incomplete information is mathematically proved to be solvable with guaranteed global convergence. However, the Bug algorithms and their variations assume a perfect capability of boundary following, which is often too ideal for a physical robot to possess.

To solve the problem of how to actually navigate a robot to follow an obstacle, Krogh and Feng [21] proposed the "subgoal method" which locates a series of local goals for navigating a point robot in polygon environments. A strategy similar to the Bug algorithms is used for switching between the two motion modes. The method sets a certain distance from edges of polygon obstacles to locate subgoals. This however is unable to guarantee that further subgoals can be generated, e.g. when the next edge is currently invisible to the robot. The limitations of the subgoal method have prevented it from being widely used for a physical robot to carry out a practical path planning task in realistic environments. Noborio *et al.* [22] attempted to solve the problem of navigating a nonholonomic mobile robot around an uncertain obstacle in a non-heuristic way by using Best-first and Depth-first algorithms for searching the state-space graph of the system's configuration space (C-space).

Boundary following is taken as a basic function or behavior by many navigation approaches [1, 3] including the Bug algorithms. Wall following, as a useful technique for following an obstacle in structured or known environments, has been extensively studied. Based on a sensing method to process the direct signal received from sonar sensors to get a better angle resolution, Yata *et al.* [23] proposed a wall following algorithm where the robot moves perpendicular to the direction of the nearest reflecting point. As it may provide a good setting for pose prediction and sensor fusion, Kalman filtering is commonly used to produce a good approximation of distance and angle from a known wall. Bemporad *et al.* [24] introduced several constraints such as velocity and maximum angle deviation from the wall to achieve better wall following.

Among reactive systems, fuzzy logic approaches are characteristic in that they deal with various situations without analytical modeling of the environment. For example, a human makes decisions based on his perceptions of obstacles, terrain and the goal, and not by mathematical analysis and models of obstacles. Fuzzy logic approaches have been used for wall following (e.g. Taheri and Sadati [25]), knowing that the natural linguistic language used in fuzzy logic would be able to capture the human sensing and intuitive reasoning for wall following. Though robust in continuously producing wall following motions upon sensory inputs, slow and jerky movements may frequently happen in such approaches.

1.2.2 Map Building and Map-aided Path Planning

One of the most important and useful features of autonomous mobile robots is their

ability to adapt themselves to operate in unstructured and unknown environments. Promising path planning and navigation algorithms require the availability of both a sufficiently reliable estimation of the current vehicle location and a sufficiently precise map of the navigation area.

Map Building and SLAM Problem

The fact that *a priori* model maps are rarely available motivates research into the problem of automatic construction of a global map from sensory information by an autonomous mobile robot. A variety of representations have been proposed in mapping research. Topological representations [26, 27] use graph-like structures to represent the environment, by associating nodes to significant places, and arcs to paths between them. Geometric approaches [28,29] use geometric primitives (line segments, circles, etc.) for representing the environment, and mapping is thus to estimate the parameters of the primitives that best fit the observations. Occupancy grid mapping algorithms [30–32] represent maps by equal-sized grids where each cell stores the probability that the corresponding place is occupied, empty or unknown. Compared with other representations, occupancy grids provide more detailed information about the environment, and can be easily updated when there is sensor input due to the probabilistic nature of grids. In indoor environments lack of global positioning system (GPS) or landmarks, and considering that a laser rangefinder is used to perceive the surroundings, occupancy grid map is used in this research.

Generally, the process of building a grid-based map is a cycle divided into two parts: "Sensor modeling" and "Updating". In "Sensor modeling", sensor (sonar, laser or vision) data collected is used as input into a sensor model which will give information about the occupancy states of the grids scanned. In the "Updating" process, the output values from "Sensor modeling" are used to update the map's corresponding grids' probability values. The cycle goes on if a further step is taken to collect more sensor data of the environment.

For "sensor modeling", most researchers interpret range sensor data in a probabilistic approach using Gaussian distribution, but different updating techniques are employed, including Bayes' Theorem [33], Dempster-Shafer theory of evidence [34,35], and fuzzy logic approaches [36, 37]. Comparison studies [38] have shown that each technique has its own pros and cons. Vector field histogram approach [7] attempted to simplify the complicated sensor model in grid-based mapping by modeling a sonar

sensor using a straight line instead of a beam-width cone.

To acquire a map, a mobile robot must possess sensors that enable it to perceive its surrounding environment. Sensors for this task include cameras, rangefinders using sonar, laser, and infrared technology, radar, tactile sensors, compasses, encoders, and global positioning systems (GPS). However, all these sensors are subject to errors, either as noise in perception (e.g., range measurements) or as noise in odometry (e.g., wheel encoders). Furthermore, motion commands issued during environment exploration carry information subject to errors, and the controls alone are therefore insufficient to determine a robot's pose. In indoor environments, without GPS, active beacons, or predefined landmarks, the robot has to rely on its on-board sensors to determine its position. Without a map the robot cannot determine its own pose due to sensor noise and motion errors, and, without knowledge about its own pose, the robot cannot build a map of the environment. Thus, mapping is often referred to as the problem of concurrent mapping and localization [39], or simultaneous localization and mapping (SLAM) [40, 41].

Kalman filters have been widely employed to estimate the map and the robot location [40–45]. Such approaches have mainly been applied where the environment contains landmarks. The resulting maps are usually described by landmarks, or other significant features in the environment. Another group of algorithms considers simultaneous localization and map building as a global optimization problem [46]. EM-based techniques [39, 47] address the correspondence problem in mapping, i.e, to determine whether sensor measurements recorded at different locations correspond to the same physical entity. They are applied to mapping large cyclic environments with highly ambiguous features. However, they require multiple passes through the entire data set and thus are not suitable for online simultaneous localization.

Scan matching [48, 49] is the process of matching two range scans obtained from a range device such as a laser rangefinder, or matching a range scan and a map. Being one commonly used method, incremental ML (maximizes likelihood) scan matching determines the most likely new pose that maximizes the consistency of the measurement with the map and the consistency of the new pose with the control action and the previous pose. Like most of other scan matching methods, it approximates alignment of two scans partially based on odometry readings. Because of its high accuracy and reliability, scan matching has been applied to many SLAM algorithms [50–52]. It should be noted that initial relative position is required by scan matching for later

pose estimation. Therefore, scan matching is most suitable for pose tracking rather than global localization.

Map-based or Map-aided Path Planning

It is essential for mobile robots to utilize partial knowledge obtained from sensory information in order to accomplish tasks such as autonomous exploration and path planning, as *a priori* model maps are not always available. As higher computational capability is available nowadays, it will be meaningful to build a map online to gain more knowledge of the environment, so that the system can enhance the probability of finding a solution if one exists, by searching the obtained partial map.

To find a path between two points in a known environment, graph search algorithms such as Dijkstras, A* algorithm [53], Bellman-Fords algorithms [54], wave front algorithm [55], visibility graph approaches [1], or Voronoi Diagram or its variations [56] can be used. A visibility graph is a set of straight lines connecting the start, the goal and obstacle vertices, where each point is connected to all viewed points without intersecting obstacles. A* and similar algorithms have emerged to solve the problem when complete information of the environment is available. The A* search algorithm is an effective heuristic improvement over the Dijkstras algorithm, and yields a better average performance when one only needs the optimum path between two grid cells in a directed graph with non-negative weights. Due to its efficiency and robustness in finding an optimal solution, the A* search algorithm is well used in gaming and path searching.

Wave front algorithm requires a grid-based map for locating a possible path. The strategy is based on the propagation of wavefronts that encodes the distance from the robot to each cell as well as the occupancy information (occupied, empty or unknown) of the cell. The planner grows a wave front from the goal position until the wave front reaches the cell that the robot is currently at: at each iteration, all cells on the wave front have the same distance from the goal cell; all cells accessed in this step are finally assigned a "plan cost" proportional to its distance from the goal. Once the evaluated costs are defined for each cell by the planner, the robot can simply follow the gradient of cells' costs until it reaches the goal.

In the field of path planning, there are a number of search-based methods [37, 57, 58] which exhaustively explore the C-space. They attempt to capture the global connectivity of a robot's free space by searching the C-space in a manner similar to

graph search in artificial intelligence. Constructing and searching the C-space for a robot of complex shape or with limited mobility (possible directions of movement) is time-consuming. This problem is partially relaxed by cell decomposition methods [59], which decompose the C-space into an array of small rectangloids (box shaped elements). As obstacles in the workspace have been explicitly taken into account, search-based methods can solve the problem of collision avoidance in the process of locating a feasible path.

Instead of assuming *a priori* information of the environment available before path planning, a couple of approaches build a global world model (e.g. an occupancy grid map) based on sensory information and use it for planning a path. Choset and Burdick [60, 61] attempted to construct Generalized Voronoi Graph (GVG) or Hierarchical GVG for incremental sensor-based path planning and exploration. GVG is efficient in regularly-shaped environments such as corridors where it is bounded by smooth walls on both sides. However, when the robot moves into vast open spaces, it would be unable to function as it cannot find an equidistant path to trace. Stentz proposed D* algorithms [62] and Focussed D* algorithms [63] to produce an optimized (in the sense of obtaining the shortest path) solution to path planning of mobile robots in known or partially known environments. As a dynamic version of the A* algorithm, D* algorithm and its variations [64] plan optimal traverses by incrementally repairing paths to the robot's state as new information is discovered. Compared to A* algorithm, they are much more complex for an implementation, and not so robust in finding an optimal solution.

One obvious advantage of "global" sensor-based path planning approaches is that, as the global world model is gradually constructed, the problem of sensor-based path planning can be converted to the problem in a known environment, and it is possible to obtain a more optimal path in the global sense. However, they have the disadvantage of being computationally expensive, since much memory and computation are involved in the process of updating the world model (which is typically done at frequencies of more than 1 HZ) or searching for a solution from it, especially when the size of the environment model is relatively large.

1.3 Motion Planning Addressing Robot Constraints

1.3.1 Robot Constraints

One class of mobile robots uses a steering mechanism for which the translational motion is independent of the orientation, i.e. they are able to move in any direction without turning their robot body [65–67]. Such robots have full omnidirectionality with simultaneous and independently controlled rotational and translational motion capabilities, which provide them with not only the advantage of continuous steering functions, but also good maneuverability. Most of conventional path planning methods mentioned previously are based on ideal movement, or omnidirecional motion of such a mobile robot.

Yet, in practice, due to the shortcomings of omni-directional robots, such as having a complex structure and relatively low payload, most of the mobile robots in use are nonholonomic ones with either a two independent driving wheels mechanism or a single wheel (or wheel pair) steering mechanism (e.g. car-like robots). Due to the existence of the nonholonomic constraint(s)[2], such robots are unable to move sideways. Moreover, various dynamic constraints, including velocity and acceleration limits, significantly restrict the motion capabilities of mobile robots. In addition, an abrupt change in the curvature of a path is not desirable, since it will cause a discontinuity in the centripetal acceleration when a robot follows the path. Among the nonholonomic systems, car vehicles are the most common ones used in practice. A car-like robot is a wheeled vehicle built based on the model of conventional cars, and is capable of autonomous motion. Its turning radius is lower bounded due to the mechanical limits on the steering angle, which imposes a curvature constraint on the path that they are able to move along. Therefore, a path for car-like robots to follow should be lower-bounded as well as continuous in its curvature.

Path planning normally considers a robot moving in a non-clear area, and therefore the robot is subject to kinematic constraints imposed by obstacles and the robot's physical dimensions. In summary, a collision-free, curvature-smooth (and curvature-bounded), and feasible (i.e. the robot dynamic constraints are satisfied) path is desirable for path planning and motion planning for nonholonomic mobile robots.

[2]Nonholonomic constraints involve the system configuration parameters and their time derivatives (velocity parameters) that are not integrable.

1.3.2 Geometric Approaches for Smooth Path Generation

Dubins [68], Reeds and Shepp [69] showed that the shortest path between two configurations for a simplified car can always be built by concatenating at most five linear or circular segments. Since then, a number of methods [70–72] compute the final path by concatenating elementary paths, such as straight-line segments and arc segments, in order for the path to meet the maximum curvature limit imposed on car-like robots. A disadvantage is that the path curvature is not continuous at the ends of the circular arc – it changes suddenly between zero and the inverse of the circular radius.

Geometric approaches use a certain complex curve, varying from cubic spirals, clothoids, harmonics function to polar polynomials, to create a smooth path with continuous position and velocity. Initiated by Kanayama and Miyake [73], arcs of clothoids, defined in the 2-dimensional Cartesian plane by means of Fresnel integrals, were used to generate a path with its curvature changing continuously and linearly along the entire path length. Fleury *et al.* [74] tried to combine clothoids with other complex curves to obtain a smooth path for a line-arc-line transition. Scheuer and Fraichard [75, 76] proposed a clothoids based steering method, CC Steer, in order to take into account the upper-bounded curvature derivative of car-like robots. The robot position is given in the form of Fresnel integrals, and can be only obtained by an integration over a period of time. Such methods are thus not suitable for path planning applications requiring information about the robot positions. Kanayama and Hartman [77] introduced a set of curves called cubic spirals which represent the curvature of the path by a second-order function of t such that the angle of the tangent is a third-order polynomial. Instead of time or path length, the curve is optimized by using the path curvature and the derivative of the curvature as criteria. Similar to the clothoids method, it does not provide a closed form solution for the robot position.

Nelson [78] suggested the use of polar polynomials to model the curve of the turns in line-arc-line transitions. A polar polynomial curve provides a closed-form expression of the coordinates, and is able to smoothly connect two line segments such that the curvature changes continuously. Pinchard et al. [79] pointed out that the maximum curvature of Nelson's curve should be less than robot's maximum admissible centrifugal acceleration divided by square of robot velocity, in order for such a curve to be used in path planning for car-like robots. However, the work didn't provide the way to explicitly compute the maximum curvature. In addition, the curve has yet

to be adapted for use with boundary conditions (e.g. transition between arcs) other than the transition between two line segments.

It is generally computationally heavy to use a complex curve for planning smooth paths, especially when the task of collision checking/test is involved. Collision checking is to verify that all configurations on a continuous path in the C-space are collision-free. Bounding-volume (BV) approaches [80] usually partition the curve by recursively bisecting and testing intermediate configurations along the curve until a collision is found or any two successive configurations are less than some pre-specified ε apart. Such approaches can either be very inefficient or have a risk of missing collision detections, depending on the size of ε. Swept-volume intersection methods [81] compute the volumes swept by the objects in the workspace and test these volumes for overlap. Exact computation of such volumes is time consuming, especially when rotations and/or geometrically complex objects are involved. Other approaches include feature-tracking methods [82], which assume that the pair of closest features between two objects in relative motion changes only at discrete points of time.

1.3.3 Dynamic Motion Planning in Velocity Space

If a series of waypoints connected with their adjacent ones are provided by a high-level path planner, the remaining problem is to generate appropriate motion commands for the robot to trace them sequentially (and avoid collision with obstacles at the same time). To accomplish this task, a number of approaches use local sensory information in a purely reactive fashion for robustness to uncertainties, and address the task of collision avoidance meanwhile. One category of such approaches is directional approach and the other is velocity space (translational-rotational velocity space) approach. Directional approach computes a direction for the robot to head in, in Cartesian space or configuration space. For instance, potential field methods [1, 3–6] use the vector summation of an attractive force representing the goal and a number of repulsive forces associated with obstacles to determine the desired robot heading. By computing a one-dimensional polar histogram from detected obstacles, vector field histogram (VFH) approach [7] improves over the traditional potential field methods, in the sense that it can achieve smoother navigation and has more chances to successfully find paths through narrow openings. Nearness Diagram algorithms [83, 84] proposed by Minguez and Montano navigate a robot reactively based on situations to simplify the difficulty of navigation in troublesome scenarios. Path

deformation method [85] iteratively deforms the current path using potential fields over the C-space in order to achieve collision-free smooth motions.

Though simple and efficient in producing a direction command for a collision-free motion, direction approach is inadequate to take the robot dynamics into account, which may result in slow or jerky movements. Velocity space approach, on the other hand, incorporates vehicle dynamics and decides both rotation and translation velocities at the same time. For example, Dynamic Window approaches [86–89] initiated by Fox *et al.* search the velocity space for a heading close to the goal direction without hitting obstacles within several command intervals. Among them, the works in [88,89] are global approaches which apply navigation function NF1 or D* algorithm for designing subgoals. Simmons's Curvature-Velocity method [90] searches the velocity space for a point that satisfies the velocity and acceleration constraints and maximizes an objective function. Though it produces reliable, smooth and speedy navigation in office environments, it has the shortcomings such as passing some collision-free paths with a high turning angle. Ko and Simmons's LCM method [91] incorporates both directional and velocity space command approaches, but it considers collision avoidance repeatedly.

Velocity space approaches typically search the velocity space for a velocity pair minimizing a single objective function. The optimized motion command found in this way may not be suitable for a real scenario, since the performance targets in the real world could be too complex to be defined as a single objective function. To find the best motion command, it is insufficient to simply define a single objective function for the constrained optimization problem [92].

1.4 Research Objectives and Scope

Based on the previous literature review, the research in this thesis looked into developing path/motion planning algorithms for autonomous mobile robots subject to various robot constraints, with the aim to achieve global convergence to the goal in unknown or partially known environments. The objectives of the research presented in this thesis were:

i) to develop a practical boundary following approach for guiding a physical robot to continuously follow an obstacle in the desired direction (i.e. on the left or right hand side) in obstacle-cluttered, complex environments. On-board sensors

such as rangefinders are used for perception of obstacles in indoor environments where path and motion planning algorithms are to be examined. The function of boundary following is used to achieve globally convergent path planning.

ii) to explore the use of complex curve to produce smooth, curvature-continuous (and curvature-upper-bounded), and feasible paths that connect two arbitrary configurations for a differential drive robot or a car-like robot. To achieve real-time path planning, a computationally efficient algorithm is to be developed for collision test of the complex curve. The path generation algorithm is to be used for a hybrid (deliberative and reactive planning) sensor based approach for smooth nonholonomic path planning.

iii) to examine a natural consideration of dynamic constraints in motion planning for a differential drive robot. The robot should be able to navigate at a relatively high speed and with robust collision avoidance capability in a dynamic, obstacle-cluttered unknown environment. In addition, convergence to each subgoal (supplied by a high-level path planner for instance) is expected for the path planning task to be accomplished.

iv) to establish a framework of hierarchical path planning capable of pose estimation and online map building, high-level path planning, and low-level motion planning. It is believed that gradual learning about the surroundings with the capability of simultaneously planning a path often results in better plans. On the other hand, deliberative planning and reactive control compliments and compensates for each other's deficiencies [93]. In this framework, the robot builds a map of the environment for better planning, and at the same time performs feasible, collision-free motions at a relatively high speed.

The research is critical because it is to deal with the difficulties of robot motion planning due to the complexities in both the environment modeling and the mobile robot itself, while global convergence to the goal is achieved in unknown environments. The study is to develop theoretical and practical algorithms for path planning and motion planning for mobile robots subject to various robot constraints and with limited information about their surroundings. One focus is to develop path planners that guaranteed global convergence to the goal, while two different methodologies (complex curve method for smooth path generation, and optimized dynamic motion

planning for relatively high speed navigation) are proposed to take care of robot dynamics and curvature constraints. Furthermore, the system builds a model of the environment incrementally such that global optimal solutions can be found.

An efficient method for pose estimation and online map building is considered for the purpose of hierarchical planning. However, the focus of the research is not to study the problem of simultaneous localization and map building itself. Designing a low-level controller to execute the motion commands for the robot to track the reference route exactly is beyond the scope of this study.

1.5 Contributions

The work presented in this thesis focused on the development of a framework for path planning and motion planing with global convergence property for mobile robots with limited information about the environment and subject to various dynamic constraints. The major contributions of this thesis are listed below:

- Kinematic and dynamic modeling of differential drive and car-like nonholonomic mobile robots. Accelerations in translation and rotation velocities in a control period have been taken into account in forward kinematics, which may help an accurate estimate of the robot poses and trajectories.

- A polar polynomial method real-time generation of a smooth, feasible path between two arbitrary robot configurations for a differential drive or car-like nonholonomic mobile robot. A velocity profile is associated with the generated path such that the robot dynamic constraints are satisfied.

  Compared with traditional complex curve approaches, the proposed method achieves efficient collision test by utilizing the particular properties of the curve. In addition, it improves over the work in [78,79] with the capabilities of checking constraints and connecting between two arbitrary robot configurations.

- A practical approach of boundary following for mobile robots by continuously locating a series of Instant Goals. Based on it, a realistic globally convergent path planner was presented for robot navigation in unstructured, complex environments. This approach is improved by considering dynamic constraints using the polar polynomial method.

One significance of this approach is that it may be used as a practical navigation function or behavior that is required by the Bug algorithms and a number of behavior-based navigation approaches.

- A practical methodology for automatic online map building by mobile robots. It fuses laser and sonar data in a selective way, in order to produce a better representation of the environment and achieve better obstacle detection.

- A hierarchical framework for incremental path planning and optimized dynamic motion planning in unknown environments. A deliberative path planner searches for an optimal path robustly with a periodically updated map, and a reactive motion planning approach generates optimized smooth, feasible robot motions for the robot to trace the path at a relatively high speed and avoid collision with obstacles effectively.

  The proposed path planner improves over the A* algorithm by handling a map containing unknown information. Accelerations in a control period are considered for the first time for dynamic motion planning such that obstacle constraints are appropriately converted to velocity limit. Compared with the velocity space approaches, multi-situations are considered in searching for an optimized velocity pair for the first time.

1.6 Thesis Outline

Following this chapter, the structure of the remaining part of this thesis is organized as follows:

Chapter 2 presents a derivation of kinematic models for differential drive and car-like nonholonomic mobile robots, and investigates how the robot dynamic constraints limit the mobility of a robot. It focus on modeling on forward kinematics, steering functions, curvature of path, and choice of reference point, and robot dynamics.

Chapter 3 proposes an Instant Goal approach for collision-free boundary following of an obstacle of arbitrary shape and globally convergent path planning in unknown environments for a holonomic robot. Collision avoidance is done reactively with a potential field approach. Based on this function of boundary following, a realistic sensor-based path planner with global convergence property is designed which uses a Bug-like strategy and makes decisions from discrete, and noisy range data.

Chapter 4 examines smooth path generation and path planning for differential drive and car-like robots. It investigates the use of polar polynomial curve to connect two arbitrary configurations smoothly while addressing the dynamic constraints. A computationally effective method is proposed for collision test of the complex curve to achieve real-time path generation. For differential drive robots, a hybrid planning approach generates a proper "Instant Goal" (and a series of deliberately plan smooth motions) and, when needed, switches to reactive planning using as a fuzzy logic controller for wall following.

Chapter 5 presents a practical method for automatical online map building. Pose estimation is achieved by applying incremental Maximum Likelihood (ML) scan matching. A selective method is proposed to fuse laser and sonar data for better obstacle detection and enhanced representation of the environment. The research also examines extracting information from grid maps to construct topological maps, which might better suit for outdoor or large-scaled environments.

Chapter 6 studies a hierarchical approach for incremental path planning and optimized dynamic motion planning in unknown environments. A* algorithm was modified to handle a map containing unknown information. A high-level planner based on it searches for an optimal (possible) path (represented by waypoints) to the goal incrementally using the periodically updated map. With the discrete path, a waypoint-directed motion command is searched in a 1D velocity space by evaluating situation-dependent objective optimization functions. To better predict the resulting robot pose and trajectories, the method considers accelerations from the current translation and rotation velocities to the commanded ones.

Chapter 7 summarizes the work presented in this thesis. It concludes this thesis and highlights the major contributions. It also discusses the limitations of this thesis and suggests future research directions that can be extended from the current research results.

Chapter 2

Modeling of Differential Drive and Car-like Nonholonomic Mobile Robots

Modeling is a prerequisite to path planning, trajectory generation and control design for mobile robots. This chapter derives kinematic models for differential drive and car-like nonholonomic mobile robots, followed by an investigation of the robot dynamic constraints, which significantly limit the mobility of the robots.

2.1 Nonholonomic Mobile Robots

2.1.1 Fundamentals

We assume that, during the motion of a robot, denoted as $\mathcal{R}$, the plane of each wheel remains vertical, and the wheel rotates about its horizontal axis. Regarding the contact between each wheel and the ground, we assume that there is only pure rolling and no slipping. This rolling condition ensures that the contact point has a zero velocity and, for any point on the wheel, the velocity has a zero component in the direction orthogonal to the wheel plane. To satisfy this assumption, the robot is assumed to move at a speed not too high in a 2D environment on a horizontal and even surface. This is to ensure that the ground friction force is great enough to prevent the robot from slipping or even turning over – the center of gravity (CG) of the robot should keep a constant distance from the ground.

As shown in Fig. 2.1, the center of a wheel, denoted by A, is a fixed point on the robot rigid frame. The pose of the wheel can be characterized by the position (ρ, θ) and the angle of the wheel plane w.r.t. the robot's body frame $\varphi(t)$. Let $\beta(t)$ denote the rotation angle of the wheel about its rotation axle. Depending on whether the orientation of its horizontal axle is fixed w.r.t. the rigid frame, the conventional wheels can be categorized into two types, i.e. fixed wheels and orientable wheels. For a *fixed wheel*, the orientation of the wheel plane is a constant angle φ. For an *orientable wheel*, the orientation of the wheel plane can be changed from time to time, that is to say, the wheel plane can rotate about a vertical (w.r.t. the ground) axis. If the vertical axis is passing through the center of the wheel, the wheel is a *centered orientable wheel*; otherwise, it is an *off-centered orientable wheel*, i.e. castor wheel.

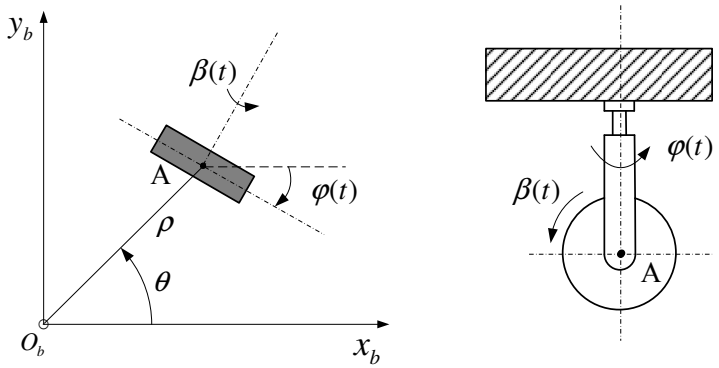


Figure 2.1: Top and side views of a wheel that is attached to robot rigid frame.

2.1.2 Kinematics of Nonholonomic Mobile Robots

There are two kinds of commonly used nonholonomic mobile robots: differential drive robots and car-like robots. A differential drive mobile robot has two rear wheels, as well as a possible castor wheel to support the robot, while a car-like robot has two rear wheels and two front wheels. The two rear wheels in both cases are fixed wheels, mounted on the same axis and aligned with the robot body. For path planning, a nonholonomic mobile robot can be modeled from a differential geometric point of view by considering only the classical hypothesis of "rolling without slipping".

The robot body frame, $o_b x_b y_b$, is established such that the origin is located at the mid point of the rear axis, W, and the x_b axis coincides with the main axis of the robot. The configuration of the robot is defined by

$$\mathbf{q} = [x \quad y \quad \vartheta]^T, \tag{2.1}$$

where (x, y) and ϑ are the position and orientation of the robot w.r.t. the global coordinate frame, respectively.

To plan a path for the robot to follow, it should be known beforehand which point on the robot body is used to track a path. One reason is that linear velocity normally varies at different points of the robot, whereas angular velocity, ω, is uniform on every part of the rigid body of the robot.

Definition 2.1. *Reference point, RP, is defined as the fixed point designated on a robot for it to track a path/route.*

Definition 2.2. *Robot velocity, v, is defined as the velocity (magnitude) at the RP for the robot to follow a path/route.*

Generally, there are both translation and rotation involved in the motion of a rigid body. Instant center (IC), or instantaneous center of zero velocity, is a point associated with a rigid body that has zero velocity at that instant. The two fixed rear wheels of a nonholonomic robot apply a constraint on the robot such that it cannot have any movement along the rear wheel axis. At each time instant, its motion can be viewed as an instantaneous rotation about the vertical axis passing through the IC, which is changing from time to time. For a nonholonomic mobile robot, the velocities consist of linear velocity and angular velocity (or translation velocity and rotation velocity), as the motion of such a robot involves both translation and rotation (straight line movement can be regarded as rotation with a zero angular velocity).

For the purpose of trajectory planning, the current state of a nonholonomic mobile robot can be expressed as a 5-tuple:

$$\mathbf{X} = [x \;\; y \;\; \vartheta \;\; v \;\; \omega]^T, \tag{2.2}$$

for the reason that the state of a mobile robot is not just determined by the robot configuration, but also the velocities of the robot.

As no slip is allowed along the rear wheel axis, the velocity at W will be zero when projected onto the rear wheel axis. This results in the following nonholonomic velocity constraint:

$$\dot{x} \sin \vartheta - \dot{y} \cos \vartheta = 0. \tag{2.3}$$

For a plane curve given by Cartesian parametric equations $x = x(t)$ and $y = y(t)$, the equation of the curvature is given by

$$\kappa = \frac{\dot{x}\ddot{y} - \ddot{x}\dot{y}}{(\dot{x}^2 + \dot{y}^2)^{\frac{3}{2}}}. \tag{2.4}$$

The longitudinal and centripetal accelerations of the robot moving along a curve are given by:

$$a_l(t) = dv/dt, \tag{2.5}$$

$$a_n(t) = \kappa v^2. \tag{2.6}$$

For path planning purpose, we are interested in the curvature of a path evaluated at the reference point (based on which a robot path is defined). Note that the curvature value should be evaluated by means of the above curvature equation and the robot's kinematic model, which may be varied depending on the chosen reference point.

Trajectories may also be represented by polynomials based on the polar coordinate system instead of the more general time-polynomials:

$$\varrho(\phi) = \sum_{i=0}^{n} a_i \phi^i, \tag{2.7}$$

where ϱ and ϕ are the polar radial and polar angle, respectively.

The tangent angle of the curve or the heading angle of the robot in polar coordinates can be expressed as

$$\gamma = \frac{\pi}{2} + \phi - \tan^{-1}\frac{\varrho'}{\varrho}, \text{ with } \varrho' = \frac{d\varrho}{d\phi}. \tag{2.8}$$

A local cartesian coordinate frame OXY is established with its origin located at the origin of the polar coordinate frame, and its X axis coinciding with the polar axis of the polar frame. Polar coordinates can be transformed into the (local) cartesian coordinates by

$$\begin{cases} X = \varrho\cos\phi \\ Y = \varrho\sin\phi. \end{cases} \tag{2.9}$$

If the cartesian coordinates (X, Y) is known, the corresponding polar coordinates can be given as

$$\begin{cases} \varrho = \sqrt{X^2 + Y^2} \\ \phi = \text{Atan2}(Y, X). \end{cases} \tag{2.10}$$

Curvature in the polar coordinate system is expressed by:

$$\kappa = \frac{\varrho^2 + 2\varrho'^2 - \varrho\varrho''}{(\varrho^2 + \varrho'^2)^{\frac{3}{2}}}, \text{with} \varrho'' = \frac{d^2\varrho}{d\phi^2}. \tag{2.11}$$

The speed of the robot moving along a curve can be given by the following in the polar coordinate system:

$$v(t) = \sqrt{\varrho^2 + \varrho'^2}\frac{d\phi}{dt}. \tag{2.12}$$

2.2 Modeling of Differential Drive Mobile Robots

2.2.1 Kinematic Modeling

A typical differential drive mobile robot has two rear wheels which are independently driven by two actuators for motion and orientation. A schematic of such a robot in rectangular shape is shown in Fig. 2.2. The dimensions of the robot are width L_1 and length L_2, and the distance between the two rear wheels is b. The body coordinate frame $o_b x_b y_b$ is established as previously stated.

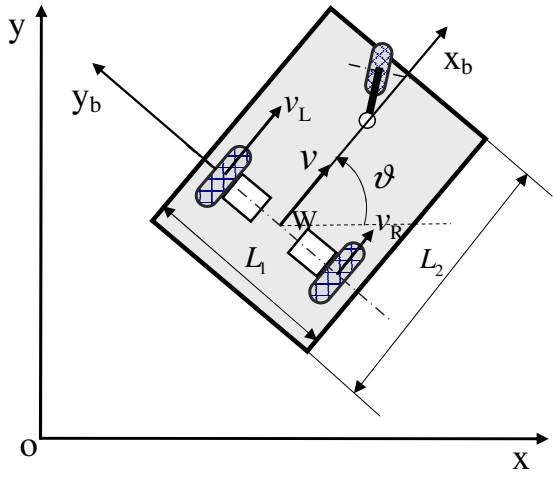


Figure 2.2: A nonholonomic robot differentially driven by two rear wheels.

From now on, (for a differential drive robot or a car-like robot) the RP is designated at the midpoint of rear-axle, W, unless otherwise stated. The robot's translation velocity v is thus defined at this point. Thus, we have

$$\dot{x} = v\cos\vartheta,\ \ \dot{y} = v\sin\vartheta,\ \ \dot{\vartheta} = \omega. \tag{2.13}$$

If the control input is defined as $\mathbf{v} = [v\ \ \omega]^T$, the kinematic model for $\mathcal{R}$ can be obtained as follows by deriving the robot configuration (2.1) to the time t:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\vartheta} \end{bmatrix} = \begin{bmatrix} \cos\vartheta & 0 \\ \sin\vartheta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}. \tag{2.14}$$

Fig. 2.3 shows the kinematics of the robot. The instant center, IC, can be obtained as the intersection of the drive axis and the line formed by the end points of the vectors

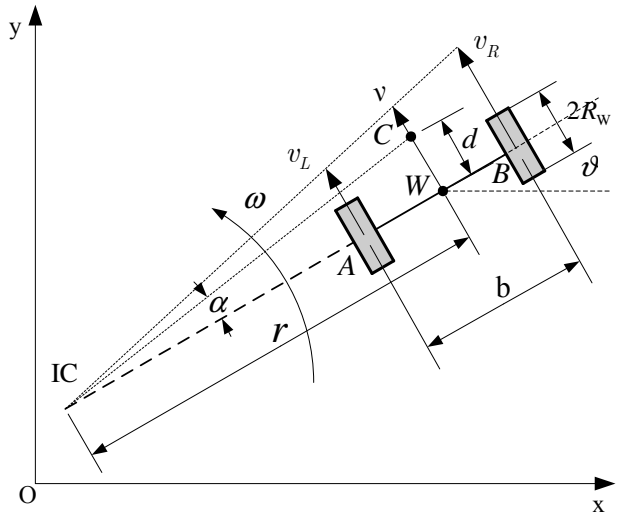


Figure 2.3: Kinematics of a differential drive mobile robot.

$\mathbf{v}_L$ and $\mathbf{v}_R$. When projected onto the line AB, the velocity vector $\mathbf{v}$ will be zero as no slip is allowed along AB.

The robot velocities at any time are related to the velocities of the left and right wheels (v_L and v_R) as follows:

$$v(t) = \frac{v_R(t) + v_L(t)}{2},\ \omega(t) = \frac{v_R(t) - v_L(t)}{b}. \tag{2.15}$$

2.2.2 Forward Kinematics

Forward kinematics is to predict the behavior of a mechanical system based on the inputs to that system, i.e. in this research how to find out the robot trajectory if the speeds are known. Let (x_0, y_0, ϑ_0) denote the initial pose of the robot. When the wheels turn at fixed velocities, the differential equations (2.13) are solvable as follows:

$$\begin{cases} x(t) = x_0 + \frac{v_0}{\omega_0}\left(\sin(\omega_0 t + \vartheta_0) - \sin\vartheta_0\right) \\ y(t) = y_0 - \frac{v_0}{\omega_0}\left(\cos(\omega_0 t + \vartheta_0) - \cos\vartheta_0\right) \\ \vartheta(t) = \vartheta_0 + \omega_0 t, \end{cases} \tag{2.16}$$

which implies that the trajectory that the robot follows is a circular path with its radius $r = v/\omega$.

Many researchers (e.g. [84, 86]) assume that the robot velocities remain constant in each control period such that the robot path will be a straight line or a circular path. However, the velocities are normally time-varying in a control period as robot are frequently accelerated or decelerated during navigation. Thus, the differential

equations (2.13) must be evaluated using numerical methods. At a relatively short interval of Δt, we may take that the wheels undergo a fixed rate of acceleration or deceleration, i.e.

$$v_L(t) = v_{L0} + a_L t, v_R(t) = v_{R0} + a_R t. \tag{2.17}$$

where v_{L0}, v_{R0} and a_L, a_R are the initial velocities and accelerations of left and right rear wheels, respectively.

Given (2.17), Eq. (2.15) can be converted to the following:

$$\begin{cases} v(t) = \frac{a_R + a_L}{2} t + \frac{v_{R0} + v_{L0}}{2} = at + v_0 \\ \omega(t) = \frac{a_R - a_L}{b} t + \frac{v_{R0} - v_{L0}}{b} = \varepsilon t + \omega_0, \end{cases} \tag{2.18}$$

where v_0, ω_0, a, ε are the initial values of translation and rotation velocities, the longitudinal acceleration, and the angular acceleration of the robot, respectively.

With Eqs. (2.18) and (2.13) and the initial condition of integral as (x_0, y_0, ϑ_0), the solution for the robot pose is:

$$\begin{cases} x(t) = \int_0^t (at + v_0) \cos\left(\frac{1}{2}\varepsilon t^2 + \omega_0 t + \vartheta_0\right)\, dt + x_0 \\ y(t) = \int_0^t (at + v_0) \sin\left(\frac{1}{2}\varepsilon t^2 + \omega_0 t + \vartheta_0\right)\, dt + y_0 \\ \vartheta(t) = \frac{1}{2}\varepsilon t^2 + \omega_0 t + \vartheta_0, \end{cases} \tag{2.19}$$

where the position can be obtained only through numerical integration method such as Simpson's Rule.

2.3 Modeling of Car-like Mobile Robots

2.3.1 Rear-wheel Drive Car-like Robots

Car-like nonholonomic robots may be categorized into rear-wheel drive, front-wheel drive, or four-wheel drive ones. Four-wheel drive car-like robots can be viewed as those having both rear-wheel drive and front-wheel drive capabilities, and being able to perform rear-wheel drive or front-wheel drive when needed. This research considers rear-wheel drive ones, as shown in Fig. 2.4. The two front wheels are centered orientable wheels, which steer the robot to the desired orientation. Either the front or rear wheels can be taken as driving wheels. L denotes the wheelbase, i.e. distance between the rotation axis of the front wheels and that of the rear wheels.

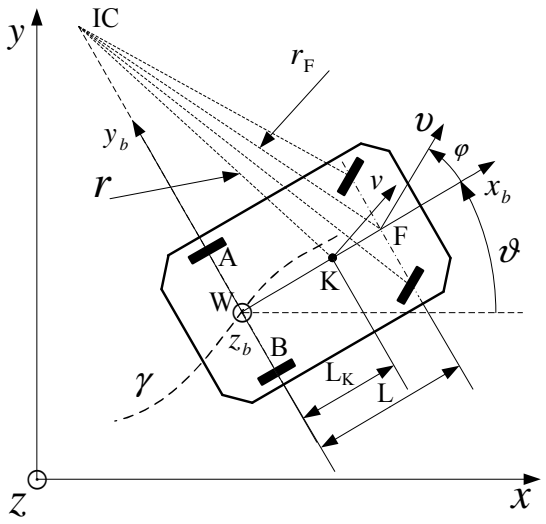


Figure 2.4: Rear-wheel drive car-like robot with the RP shifted to K.

The configuration of a car-like robot can be defined by the 4-tuple

$$[x \quad y \quad \vartheta \quad \varphi]^T, \tag{2.20}$$

where φ is the steering angle of the front wheels w.r.t. the body frame. φ is mechanically limited and bounded by an angle, $\varphi_{\max}$, that is less than 90° in practice:

$$|\varphi| \leq \varphi_{\max}. \tag{2.21}$$

Similar to the derivation of (2.3), the component of velocity normal to the steering wheels should be zero, from which we have

$$\dot{x}\sin(\vartheta+\varphi) - \dot{y}\cos(\vartheta+\varphi) - L\dot{\vartheta}\cos\varphi = 0. \tag{2.22}$$

Front-wheel velocity $\mathcal{V}$ is defined as the velocity at F, the mid point of the front axle of the car-like robot. According to relative motion principles, the motion of point F is relative to that of point W by $L\dot{\vartheta} = \mathcal{V}\sin\varphi$. Therefore, the vehicle heading ϑ can be integrated from the following equation:

$$\dot{\vartheta} = \mathcal{V}\sin\varphi/L. \tag{2.23}$$

A car-like robot is not always turning around a fixed IC and φ is not always constant – the robot can be steered to a different robot velocity. However, the front velocity at that time can be considered as a constant, i.e. $d\mathcal{V}/dt = 0$ for a very short duration.

2.3.2 Shifted Reference Point

Although the RP may be somewhat arbitrarily designated, the choice of RP affects the kinematics of a robot, the space that the robot is able to cover, and the steering functions. These, in turn, will influence path planning for a mobile robot.

In this sub section, the RP is shifted forward from W along the main axis by a distance L_{K} to point K, as shown in Fig. 2.4. In addition to that the vehicle heading ϑ can be integrated using the relationship in Eq. (2.23), the derivatives of the position at this RP are related to the front-wheel velocity and steering angle pair $(\mathcal{V}, \varphi)$ by the following kinematic equations:

$$\begin{cases} \dot{x}_{\mathrm{K}} = \mathcal{V}(\cos\vartheta\cos\varphi - \sin\vartheta\sin\varphi L_{\mathrm{K}}/L) \\ \dot{y}_{\mathrm{K}} = \mathcal{V}(\sin\vartheta\cos\varphi + \cos\vartheta\sin\varphi L_{\mathrm{K}}/L). \end{cases} \tag{2.24}$$

As v is now defined at K and thus $v = \sqrt{\dot{x}_{\mathrm{K}}^2 + \dot{y}_{\mathrm{K}}^2}$ holds, using Eq. (2.24), the relationship between v and $\mathcal{V}$ can be derived as follows:

$$\mathcal{V} = v/\sqrt{\cos^2\varphi + (L_{\mathrm{K}}^2/L^2)\sin^2\varphi}. \tag{2.25}$$

The values of $\ddot{x}_{\mathrm{K}}$ and $\ddot{y}_{\mathrm{K}}$ may be obtained by differentiating Eq. (2.24), with in mind that $d\mathcal{V}/dt = 0$, ϑ and φ are functions of t. Knowing that $v = \sqrt{\dot{x}_{\mathrm{K}}^2 + \dot{y}_{\mathrm{K}}^2}$ and Eq. (2.23) holds, by substituting the obtained values of $\ddot{x}_{\mathrm{K}}$ and $\ddot{y}_{\mathrm{K}}$, and Eq. (2.24) into the curvature equation (2.4), we have the following equation for the curvature:

$$\kappa_{\mathrm{K}} = \frac{\mathcal{V}^3}{v^3}\left(\frac{L_{\mathrm{K}}\dot{\varphi}}{L\mathcal{V}} + \frac{\sin\varphi}{L^3}\left(L^2\cos^2\varphi + L_{\mathrm{K}}^2\sin^2\varphi\right)\right). \tag{2.26}$$

With Eq. (2.25), the relationship between v and $\mathcal{V}$, Eq. (2.26) can be simplified as follows:

$$\kappa_{\mathrm{K}} = \frac{LL_{\mathrm{K}}}{v}\frac{\dot{\varphi}}{L^2\cos^2\varphi + L_{\mathrm{K}}^2\sin^2\varphi} + \frac{\sin\varphi}{\sqrt{L^2\cos^2\varphi + L_{\mathrm{K}}^2\sin^2\varphi}}. \tag{2.27}$$

The curvature (2.27) is a function not only of φ and v but also of $\dot{\varphi}$, which is correlated to the "flatness" properties [94] of the car-like model. As shown in Eq. (2.27), the function of steering angle, φ, is not easy be derived if the path (and thus the values of v and κ_{K}) is given.

When the front (steering) wheel is selected as the RP, i.e. $L_{\mathrm{K}} = L$, the derivatives of the state at this RP are related to the pair $(\mathcal{V}, \varphi)$ by the following kinematic equations:

$$\begin{cases} \dot{x}_{\mathrm{F}} = \mathcal{V}\cos(\vartheta + \varphi) \\ \dot{y}_{\mathrm{F}} = \mathcal{V}\sin(\vartheta + \varphi). \end{cases} \tag{2.28}$$

Now v is equal to $\mathcal{V}$, i.e. $\mathcal{V} = v$. The curvature of the path is then given by

$$\kappa_{\mathrm{F}} = \frac{\dot{\varphi}}{v} + \frac{\sin\varphi}{L}. \tag{2.29}$$

If the boundary condition is known (e.g. $\varphi(0)$), the nonlinear differential equation (2.29) can be integrated numerically to find the equation for the steering angle φ, which however is still unable to be written into an explicit form.

As pointed out by [95], the discontinuities in the steering functions when tracking a non-continuous curve can be eliminated by shifting the RP. When there is a non-continuous change in the curvature κ_{F}, the solution to the nonlinear differential equation (2.29) is not necessarily non-continuous (as $\dot{\varphi}$ could be non-continuous in Eq. (2.29) while φ is continuous). Of course, it can be seen from Eq. (2.25) that there will be no discontinuity on the front-wheel velocity $\mathcal{V}$ either.

The vehicle heading is now not aligned with the path after the RP starts to track the arc, and until when a distance L along the new line segment has been traversed by the RP. That is to say, a certain amount of distance is required for the robot to properly align its orientation after following a turn. Therefore, path planning may become more complex with this setting of RP.

2.3.3 Reference Point Selected at W

In the remaining part of this thesis, the RP is chosen to be W (as normally is), unless otherwise stated. v is now the velocity at W. Therefore, we have

$$v = \mathcal{V}\cos\varphi. \tag{2.30}$$

Knowing that $\dot{x} = v\cos\vartheta,\ \dot{y} = v\sin\vartheta$, we have

$$\begin{cases} \dot{x} = \mathcal{V}\cos\varphi\cos\vartheta \\ \dot{y} = \mathcal{V}\cos\varphi\sin\vartheta. \end{cases} \tag{2.31}$$

If the control input is defined as $\mathbf{v} = [v \quad \dot{\varphi}]^T$, the kinematic model for rear-wheel drive car-like robots can be derived from Eqs. (2.31), (2.23) and (2.30) as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\vartheta} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \cos\vartheta \\ \sin\vartheta \\ \tan\varphi/L \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \dot{\varphi}, \tag{2.32}$$

where the first vector field has a discontinuity at $\varphi = \pm\pi/2$ (i.e. the front wheels are normal to the main axis of the vehicle). However, this singularity seldom occurs in practical cases, where the range of the steering angle φ is restricted.

Instead of the model (2.32) that is convenient for control, we are more interested in obtaining the steering functions for the purpose of path planning.

The values of $\ddot{x}$ and $\ddot{y}$ may be obtained by differentiating Eq. (2.31) while noticing that $d\mathcal{V}/dt = 0$, ϑ and φ are functions of t. Knowing that $v = \sqrt{\dot{x}^2 + \dot{y}^2}$ and Eqs. (2.23) and (2.30) hold, by substituting the obtained values of $\ddot{x}$ and $\ddot{y}$, and Eq. (2.31) into the curvature equation (2.4), we have

$$\kappa = \dot{\vartheta}/v = \omega/v = \tan\varphi/L, \tag{2.33}$$

which shows that the curvature can be obtained from the values of the velocity at the RP and the angular velocity of the robot.

The steering functions to derive the front-wheel velocity and steering angle are then given by:

$$\varphi = \tan^{-1}(L\kappa), \tag{2.34}$$

$$\mathcal{V} = v/\cos\varphi, \tag{2.35}$$

which implies that there will be an abrupt change in the steering angle if there is a non-continuous change in the curvature.

The curvature κ at the RP is now equal to the inverse of r, the distance of the IC from W, i.e.

$$r = 1/\kappa. \tag{2.36}$$

Let $r_{\min}$ denote the minimum turning radius of a car-like robot. From Eqs. (2.21), (2.33) and (2.36), we know that the following curvature constraint is imposed on the robot's path:

$$\kappa \leq 1/r_{\min} = \tan\varphi_{\max}/L. \tag{2.37}$$

Fig. 2.5 shows the trajectories of the RP and the steering wheel, when the robot follows a path consisting of two line segments and one arc with radius r_C. In this case, the circular center of the arc, Ω_C, is at the same time the IC of the robot motion during its moving along the arc. This path is smooth in the sense that there is no abrupt change of direction along the path.

As the RP moves along line segment $\overrightarrow{AB}$ or $\overrightarrow{CD}$, the front (steering) wheel moves along the line segment or its extension. When the RP moves along the arc segment,

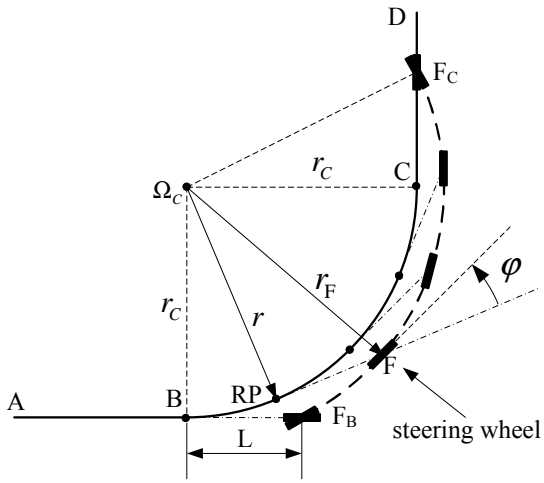


Figure 2.5: Trajectories of the center of the rear wheel axis and that of the steering wheels when following a path consisting of both line and arc segments.

the robot rigid body is rotating around the IC, i.e. Ω_C. When the RP is at B or C, the front wheel will be located at $\mathrm{F_B}$ or $\mathrm{F_C}$, respectively. Thus, as the RP moves along the arc segment, the front wheel moves along the dashed arc segment of radius r_{F} and starting from $\mathrm{F_B}$ and ending at $\mathrm{F_C}$. When the front wheel completes the dashed arc, the vehicle heading is aligned with the line segment $\overrightarrow{CD}$. The vehicle heading is continuously aligned with the line segment, or the tangent to the arc segment, whichever is being tracked.

As can be seen from Fig. 2.5, there will be an abrupt change (between zero and $\tan^{-1}(L/r_C)$) in the steering angle when the RP is at a line-arc transition. This fact also can be deduced from the steering functions (2.34) and (2.35): as the curvature value of the path is zero on line segments, and $1/r_C$ on arc segments, both the steering angle and the steering speed will have a discontinuity when the RP is at a line-arc transition.

2.4 Robot Dynamic Constraints

Fig. 2.6 shows the dynamic model of a nonholonomic mobile robot. The x_b axis of the body coordinate frame coincides with the vector tangent to the path γ at W. The z_b axis is chosen such that (x_b, y_b, z_b) forms a right-handed coordinate system.

According to Newton's first law of motion, a moving body travels along a straight path with a constant speed unless it is acted on by an external force, say F_{x_b}, to make it accelerate or decelerate. For a circular motion to occur, there must be a force F_{y_b}

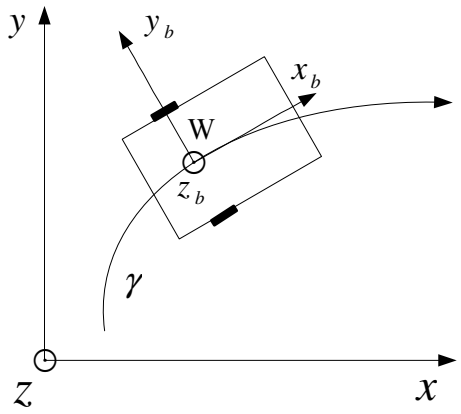


Figure 2.6: Frame attached to a nonholonomic mobile robot.

acting on a body, pushing it toward the center of the circular path. The magnitude of this centripetal force is equal to the mass m of the body times its velocity squared and divided by the turning radius. The motion along the path γ obeys Newtonian dynamics:

$$\begin{cases} ma_l & = F_{x_b} \\ m\kappa v^2 & = F_{y_b} \\ 0 & = F_{z_b} - mg, \end{cases} \tag{2.38}$$

where g is the constant gravitational acceleration, and F_{z_b} is the normal force by the ground.

The component of $\overrightarrow{F}$ in the plane represents the friction applied from the ground to the wheels. Since no sliding is allowed, there is a friction constraint as follows:

$$\sqrt{F_{x_b}^2 + F_{y_b}^2} \leq \mu F_{z_b}, \tag{2.39}$$

where μ is the friction coefficient between the wheel tires and the ground.

Substituting the results of Eq. (2.38) into Eq. (2.39), we have

$$|a_l| \leq \sqrt{\mu^2 g^2 - \kappa^2 v^4}. \tag{2.40}$$

There is also a bound on the torque (or force) $F_{\max}$ applied by the engine on the wheels. Together with the friction constraint (2.40), it yields the following feasible range of the acceleration:

$$|a_l| \leq \min(F_{\max}/m, \sqrt{\mu^2 g^2 - \kappa^2 v^4}). \tag{2.41}$$

Besides the maximum velocity $v_{\max}$ determined by the rotational speed of the wheels and the air friction force, the velocity v is constrained by the requirement

that the square root in Eq. (2.41) should be nonnegative. The feasible range of the velocity is then given by

$$|v| \leq \min(v_{\max}, \sqrt{\mu g/\kappa}). \tag{2.42}$$

The maximum angular velocity is also limited due to the actuators' capability:

$$|\omega| \leq \omega_{\max}. \tag{2.43}$$

For a car-like robot, due to the actuators' limits, there is a constraint on the steering velocity, i.e. the changing rate of the steering angle:

$$|\dot{\varphi}| \leq \dot{\varphi}_{\max}. \tag{2.44}$$

2.5 Summary

The kinematic models of differential drive and car-like nonholonomic mobile robots have been derived in this chapter. In addition, the robot dynamic constraints, which significantly influence a robot's motion capabilities, were developed. Compared with the modeling of wheeled mobile robots [96, 97] or robot carts [95], this chapter is focused more on modeling issues related to forward kinematics, steering functions, curvature of path, and choice of reference point and robot dynamics, which are directly correlated with the research work of this thesis.

The main contribution of this chapter is its investigation of kinematic modeling, steering functions and robot dynamic constraints of differential drive and car-like robots, which is a basis for subsequent path planning and motion planning. Accelerations in translation and rotation velocities in a control period have been taken into account in the study forward kinematics, which may help to estimate the robot poses and trajectories more accurately. In addition, it presents new understandings about the choice of reference point for a robot to track a nominal path as well as its influence on path planning and motion planning.

Chapter 3

Boundary Following and Convergent Path Planning Using Instant Goals

In this chapter, Instant Goal approach is proposed for collision-free boundary following of an obstacle of arbitrary shape and globally convergent path planning in unknown environments. Firstly, for effective knowledge representation and manipulation, a vector representation is presented to represent the local environment based on sensory range data. Next, the concept of Instant Goal is introduced for a holonomic robot to perform boundary following in a "natural" human-like manner. The robot moves "forward" along the boundary, even if the obstacle is of arbitrary shape and other obstacles are present nearby. Then, a potential field based reactive collision avoidance is performed simultaneously and, when needed, is efficiently incorporated in. Based on the boundary following function, a realistic sensor-based path planner with global convergence property is designed for a robot that is able to acquire discrete, and noisy range data.

3.1 Introduction

Considering the fact that any smooth boundary can be partitioned as piecewise linear, based on previous work on wall following [23, 24], the related problems of boundary following can be solved without any difficulty by traveling in a parallel direction to the curve tangent. However, a robot is usually only able to have a discrete

approximation of its surroundings, which makes the problem difficult. Moreover, with the presence of nearby obstacles which may disturb the moving direction of the robot, the problem is not just following a curve while keeping a constant distance from it, but also involves making decisions as to which obstacle to follow. There is no ready solution for a robot to follow obstacles of complex shape – the Bug algorithms and their variations thus make an ideal assumption about the capability of boundary following.

This chapter presents an Instant Goal approach which allows a physical robot equipped with a rangefinder to follow an obstacle of arbitrary shape. "Instant Goal" is a local target specially designed for action planning based on sensory input. Instant Goal Driven method was first presented in [98] where Instant Goals are generated at each time instant for behavior-based navigation. In this research, the robot is enabled to move *forward* along the boundary of an obstacle without collision in an obstacle cluttered environment. A vector representation saving much on memory space is used to represent the local environment sensed by a rangefinder. The search range to determine an Instant Goal is restricted to a certain scope computed based on the previous navigation status and the current range data input, to ensure that boundary following is performed in the desired direction. The concept of "safe path" is used to distinguish the "disturbing" obstacles from the desired one such that Instant Goals can be determined under any complex obstacle condition.

Based on the boundary following function, a practical convergent path planner is presented for mobile robot navigation without first building a global map of the environment. It uses a Bug-like strategy to switch between the two motion modes in order to achieve global convergence of path planning. A special leave condition is proposed for the robot to decide whether it should leave the currently followed obstacle and switch back to move toward the goal. Rather than assuming the robot is able to follow an obstacle, the path planner is based on the Instant Goal approach, and does not require a range sensor to have a perfect sensing ability.

3.2 Representation and Modeling of Local Environment

Many approaches of modeling have been suggested in the literature for localization, map building and path planning for robotic systems. In reactive motion

planning, representations of the local environment should aid motion determination or behavior generation and satisfy the real-time requirement. The robot has limited memory that only allows it to store a limited number of important points, which makes it often insufficient, for example, for storing incremental maps which may be very large. In this section, a simple vector is used to represent the local environment and a simple way is proposed to model the sensed environment based on range data.

3.2.1 Vector Representation of Local Environment

On-line sensors are utilized by the robot to sense the environment where the robot navigates. For an indoor environment, rangefinders such as ultrasonics and laser scanners are often used because they can directly measure the ranges of obstacles. In this research, a laser scanner is used, considering that it may provide much better angular resolution and there are less spurious readings. It is assumed that the laser scanner is able to provide N_s equally spaced angular readings for a rotation of 360°. As shown in Fig. 3.1, the detectable area of the laser sensor is denoted by the dashed circle of radius R_d centered at O_b, while the sensed free-space is denoted by the region encircled by solid dashed lines.

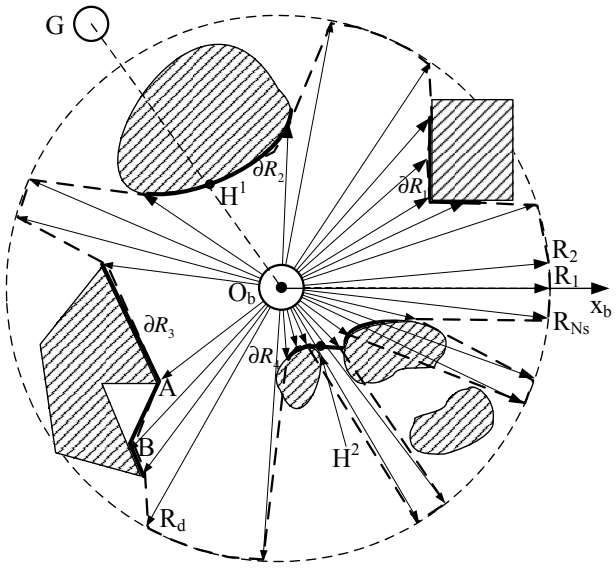


Figure 3.1: Obstacles sensed by a laser rangefinder attached to a circular robot.

A holonomic robot is assumed to be used in this chapter. Define effective size of a robot as the diameter (denoted by $2R_{\text{rob}}$) of the robot body which can be the actual diameter for a circular robot or the diagonal length for a rectangular robot.

The N_s beams of range readings are indexed in a counterclockwise (CCW) manner with the direction of the first beam coinciding with the x_b axis (or the orientation of the right side of $\mathcal{R}$ if the angular range is 180°). The 2D rangefinder information obtained in the j^{th} beam direction is represented as pairs $p_j = (\rho_j, \theta_j), j \in [1, N_s]$, in polar coordinates with respect to the sensor coordinate frame attached to the laser scanner, where ρ_j is the measured distance of the detected object, and θ_j is the measured azimuth angle between the direction of the beam and the x axis of the sensor's frame. If the relationship between the index and the beam angle is known for each beam, the measured point can be represented in a more convenient form:

$$p_j = (\rho_j, j), j \in [1, N_s]. \tag{3.1}$$

A point on an object (an obstacle or the goal) is *detectable* if and only if it is within the sensor range. A point is *visible* if and only if it is detectable and its viewing line (the straight line segment from P to the point) does not intersect any other object(s). Furthermore, it is known that, if a point is visible from point P, it is also visible from any point between it and P.

Grids are often used to represent the navigable space around the robot due to their simplicity [99]. Grid representations are arbitrarily tessellated regions surrounding the robot and can vary in resolution, shape, and uniformity. The simplest version involves a two-dimensional array of cells, where each cell in the array corresponds to a square of fixed size in the region being mapped. Radial sector grid representations [100, 101] have also been used for motion planning. One drawback of grid representations is that if the sensed area is large, they may consume a large amount of memory resources and require a long computation time to generate appropriate motion behaviors. Another disadvantage is that the distances from the robot to the obstacles, which are often measurements obtained directly from onboard sensors and the input to a motion generator, are hidden in the grid representations and need to be inferred from the indices of the grids. Moreover, at each instant, the information of the obstacle(s) hidden behind the detectable ones is not needed for the purpose of local navigation.

Taking these into account, a simple vector proposed in the work of [98] is used to represent the local environment:

$$\mathbf{R} = [R_1, ...R_j, ...R_{N_s}]^T \in \mathbb{R}^{\mathrm{N_s}}, \tag{3.2}$$

where R_j $(j = 1, 2, \ldots, N_s)$ is the measured distance between the robot and the obstacle detected in the j^{th} direction. When no obstacle is detected, R_j is set as R_d.

As shown in Fig. 3.2, the vector map saves much space compared to a grid map or a radial sector grid map. For instance, if the beam number is eight, the vector map can be described by a vector $\mathbf{R} = [R_1, R_2, R_d, R_4, R_5, R_d, R_7, R_8]^T$. To represent the same local space, the radial sector grid map will use a $8 \times N_d$ matrix in the following form, where N_d is the grid dimension along the range:

$$M^T = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \times & 0 & 1 & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \times & 0 & 1 & \times & 0 & \times & \times \end{bmatrix},$$

where a zero value, a non-zero value or "×" indicates that the corresponding space is free, occupied, or undetermined, respectively.

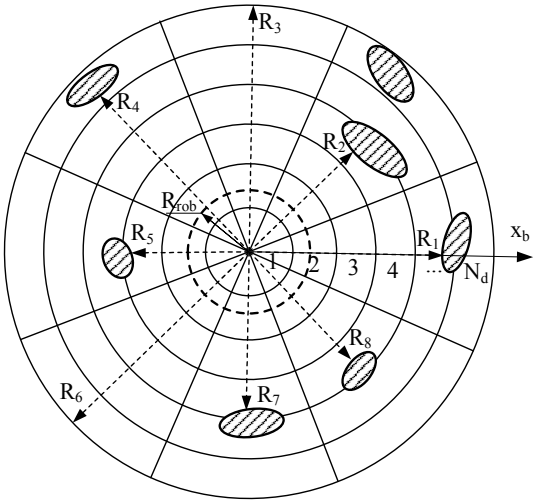


Figure 3.2: Representations of local environment sensed by a laser rangefinder.

In actual implementations, the laser rangefinder is often mounted at the center of the robot. In that case, the sensor reference coincides with the body coordinate frame, and the local coordinates of each obstacle point can be easily converted from the expression of Eq. (3.2). Let R_{G} denote the distance of the goal from the robot, and j_{G} denote the index of the beam nearest to θ_{G}, the direction of the goal. When the goal is detected by the sensor, the area behind the goal will be of no interest to the robot. Taken this into account, a modified version of the vector representation, $\hat{\mathbf{R}} \in \mathbb{R}^{\mathrm{N_s}}$, is used to describe the local environment by defining the j^{th} element of $\hat{\mathbf{R}}$

as follows:

$$\hat{R}_j = \begin{cases} R_j, & j \in (1, 2, \ldots, N_s),\ j \neq j_{\mathrm{G}} \\ \min(R_j, R_{\mathrm{G}}), & j = j_{\mathrm{G}}. \end{cases} \tag{3.3}$$

The vector representation is natural in the sense that it resembles the distribution of sensory data. Such a representation is suitable for behavior generation and obstacle avoidance of a robot with a rangefinder. In potential field methods, for example, the repulsive forces associated with the individual known obstacles can be directly derived from the vector representation.

Remark 3.1. *The vector representation of the local environment has the following advantages: i) it can be directly constructed based on the range measurements from range sensors such as sonar array, laser rangefinder or infrared, etc.; ii) compared with grid representations, it saves greatly on memory space, considering that it is a vector of dimension N_s rather than a $N_s \times N_d$ matrix of the same element size; and iii) it is able to represent the local environment sensed by a rangefinder, and is suitable for behavior generation and obstacle avoidance of a robot with a rangefinder.*

3.2.2 Modeling of Sensed Environment

Grouping of Object Points into Obstacles

The above representation of the local environment has not taken into account the uncertainties of measurements, which however should be properly incorporated in for safe navigation. The distance uncertainty of range sensors is often a function of range and identified by *range error variance* σ_ρ^2. For example in the work of [43], a fit for the range error curve of a laser scanner is given as $\sigma_\rho = 9.6 \times 10^{-7}\rho^2 - 5.6 \times 10^{-4}\rho + 21.213\ \ (mm)$. Maximum range error is denoted by σ_{Rd} since it typically occurs when the value of a range measurement is around R_d.

A set of object positions can be obtained from one laser scan. If a range value is less than the maximum limit, it implies that an obstacle has been detected. The scanned sequence of range, $\mathcal{S} = \{p_j | j = 1, \ldots, N_s\}$, can be divided into *obstacle point set* and *non-obstacle point set* as follows:

$$\bigcup_{k=1}^{n_{\mathcal{O}}} \mathcal{O}_k = \{p_j | p_j \in \mathcal{S}, \rho_j < R_d - \sigma_{R_d}\}, \tag{3.4}$$

$$\bar{\mathcal{O}} = \{p_j | p_j \in \mathcal{S}, \rho_j \geq R_d - \sigma_{R_d}\}, \tag{3.5}$$

where $n_{\mathcal{O}}$ is the number of obstacles. With this partition method of measured points in mind, the sensed free-space around $\mathcal{R}$ in Fig. 3.1 can be denoted by the region encircled by solid dashed lines.

As noticed from Fig. 3.1, these obstacle points can be grouped into several obstacles at a glance. Using the discontinuity property between adjacent observed points, the obstacle points may be divided into several different obstacles using a simple criterion that is based on constant thresholds such as:

$$\|p_j - p_{j+1}\| < \varepsilon.$$

For example, an obstacle can be regarded as connected by the criterion that the distance between the associated n adjacent obstacle points is smaller than the effective size of the robot, i.e.,

$$\partial R = \{p_j | \sqrt{\hat{R}_j^2 + R_{j+1}^2 - 2\hat{R}_j\hat{R}_{j+1}\cos(\tfrac{2\pi}{N_s})} < 2R_{\text{rob}}, j = j_1, \ldots, j_1 + n - 1\}. \quad (3.6)$$

However, such approaches may be unable to cope with range data, because the distance between two measured points depends on both the angle formed by the beams and the normal vector of the surface being scanned. For example, although the distance of two adjacent obstacle points A and B in Fig. 3.1 is greater than $\mathcal{R}$'s effective size, for navigation purposes, they should be regarded as two points belonging to the same obstacle.

Range-based Straight Path and Obstacle/Angle Scope

The term *path* refers to the shape of a motion, that is, the shape of the curve in the robot's configuration space. In this research, "straight path segment" is used to determine if it is safe for $\mathcal{R}$ to move from its current position in a certain direction.

Straight path segment $\gamma_{AB}(r)$ is defined as the segment consisting of a straight line segment $\overrightarrow{AB}$ with a thickness radius r and the semi-disk centered at B with a radius r which is not overlapped with the "thick" line segment $\overrightarrow{AB}$. As shown in Fig. 3.3, $\gamma_{AB}(r)$ is the space inside the thick solid lines $\overline{b_2a_2}$, $\overline{a_2a_1}$, $\overline{a_1b_1}$ and arc $\overline{b_1b_2}$ and it does not equal to $\gamma_{BA}(r)$. The *length* of $\gamma_{AB}(r)$, measured by $|AB|$, is denoted as ℓ. In the context of discrete range data, $\gamma_{j,\ell}(r)$ denotes the straight path segment of length ℓ in the j^{th} beam direction.

Definition 3.1. *The straight path segment $\gamma_{AB}(r)$ is said to be safe for the robot with an effective radius R_{rob} to move from its current position A to its destination B, if*

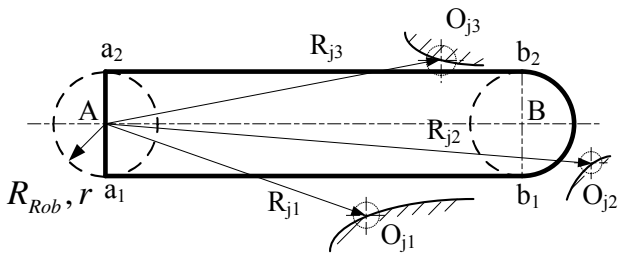


Figure 3.3: Straight path segment: the space inside thick solid lines.

and only if $r \geq R_{\text{rob}}$ *holds and all the obstacle points are not in the path segment. In addition, a function SafeSP*$(\gamma_{AB}(r))$ *is defined to check if the straight path* $\gamma_{AB}(r)$ *is safe for the robot.*

In Fig. 3.3, the measured position value of each obstacle point is denoted by the center of a small dashed circle and, due to the uncertainties in range sensing, the actual position of the point is however only known to be confined to the circle. Suppose that a robot of effective radius $R_{\text{rob}} = r$ moves from A to B, $\gamma_{AB}(r)$ is safe for obstacle points O_{j_1} and O_{j_2}. However, when uncertainties of the range sensing are considered, $\gamma_{AB}(r)$ is not safe for obstacle point O_{j_3}.

There exists a safe straight path from point A to point B for the robot, if and only if the straight path segment $\gamma_{AB}(R_{\text{rob}} + \sigma_{R_d})$ is safe for the robot to move from point A to point B. If any of the beams on the right-hand side of $\overline{a_1 a_2}$ does not intersect line segments $\overline{b_2 a_2}$, $\overline{a_1 b_1}$ or arc $b_1 b_2$, there is no safe path from point A to point B for the robot.

As shown in Fig. 3.4, *left obstacle range* $\partial R_C^L(A, B)$ is defined as the continuous obstacle boundary (denoted as ∂R_1) from the start visible point A to the end visible point B on the left (CCW) direction w.r.t. the start line (here is $\overrightarrow{CA}$) direction. *Left angle range* $\partial\theta_C^L(A, B)$ is defined as the corresponding angle range. *Right obstacle range* $\partial R_C^R(A, B)$ and *right angle range* $\partial\theta_C^R(A, B)$ can be similarly defined. Note that $\partial R_C^L(A, B) \neq \partial R_C^R(A, B)$, and $\partial\theta_C^L(A, B) \neq \partial\theta_C^R(A, B)$.

As such, if $\overrightarrow{CA}$ and $\overrightarrow{CB}$ correspond to the ${j_1}^{th}$ and ${j_2}^{th}$ range readings respectively, left obstacle range $\partial R_C^L(A, B)$ can be denoted as $\partial R_C^L(j_1, j_2)$ and left angle range can be denoted as $\partial\theta_C^L(j_1, j_2)$.

3.3 Boundary Following through Instant Goals

In this section, a range-based method is proposed to determine a series of Instant

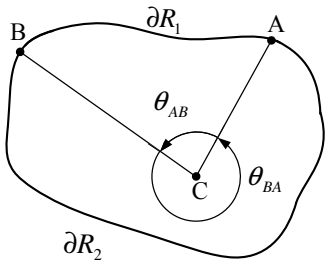


Figure 3.4: Illustration of directional obstacle range and angle range.

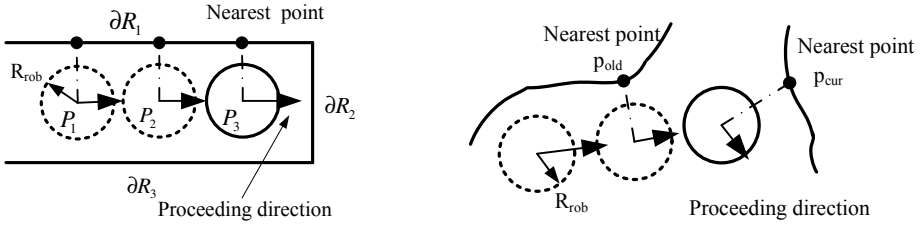
Goals which guide the robot to follow the boundary of an obstacle of arbitrary shape.

3.3.1 New Strategy of Boundary Following

The basic idea of conventional wall following algorithms is to maintain a set of (fixed) distance to a wall using a series of range measurements. The surface is followed by moving in a perpendicular direction to the nearest point on an obstacle. Using this strategy, a convex corner, concave corner or even a curve surface can be followed [23]. Such algorithms assume a sufficiently fast measurement (to obtain the nearest reflecting point) relative to the velocity of the robot and a capability of accurately measuring the direction to the reflecting point.

However, the reliance on measurement of the nearest reflecting point to determine the motion direction not only requires a very good signal processing, but may also bring about improper behavior when the obstacle consists of some special shapes, or if there are other obstacles detected very close to the one currently followed. In Fig. 3.5(a), ideally the robot's proceeding direction should be kept as right when following boundaries ∂R_1 and ∂R_2, and left when following ∂R_3. However, it can be seen that, due to uncertainties, the directions obtained have a 50% probability of turning out to be incorrect. In an obstacle cluttered environment as in Fig. 3.5(b), the robot may unexpectedly turn to follow other obstacle rather than the original one.

Instant Goal approach is proposed for a holonomic robot to follow an obstacle in complex, obstacle-cluttered environments. At each step, an *Instant Goal* (IG for short), a point serving as a temporary goal for the robot to reach, is computed. The robot will move *forward* in the sense of following the obstacle, by restricting the search range of each IG to a special scope of the boundary. In the context of using a laser rangefinder, suppose that an IG is in the ${j_{\rm IG}}^{th}$ beam direction and of a distance $r_{\rm IG}$ (not necessarily equals to $\hat{R}_{j_{\rm IG}}$) from the laser sensor. If the sensor is mounted at the



(a) A U-shaped obstacle. (b) Cluttered obstacles.

Figure 3.5: Robot may exhibit an improper behavior during wall following.

center of the robot, the IG can be expressed in the polar coordinate form w.r.t. the robot as follows:

$$(r_{\mathrm{IG}}, \theta_{\mathrm{IG}}) = (r_{\mathrm{IG}}, \frac{2\pi}{N_s}(j_{\mathrm{IG}} - 1)). \tag{3.7}$$

In the boundary following mode, until the IG is reached or need to be revised, the robot will keep moving toward it with its orientation as follows:

$$\vartheta = \angle\overrightarrow{P_t P_{\mathrm{IG}}}. \tag{3.8}$$

Start, S_i, is defined as the location at which the robot decides either to follow an obstacle or to move directly toward the goal, where $i = 1, 2, \cdots$ is the index of the obstacle met by the robot. The robot position at which an IG is determined is named as *Instant Start*, $S_{i,k}$, where $k = 1, 2, \cdots$ is the index of *Instant Starts* during the following of the i^{th} obstacle. Similarly, an IG is denoted by $G_{i,k}$. Algorithm 1 briefly describes the procedure of following an obstacle.

Algorithm 1 Following An Obstacle.

```
1: S_i ← P_t, k ← 1
2: Assign a following direction
3: while the whole boundary is not covered do
4:     S_{i,k} ← P_t                          ▷ Set current Instant Start
5:     Compute search range
6:     Determine Instant Goal G_{i,k}
7:     while G_{i,k} is not reached do
8:         Move toward G_{i,k} in P_tG_{i,k}→ direction
9:         if Instant Goal need to be revised then
10:            Exit While
11:        end if
12:    end while
13:    k ← k + 1
14: end while
```

When $\mathcal{R}$ is moving toward an IG, only the range data in "front" of $\mathcal{R}$ are considered to be "concerned" for the purpose of obstacle avoidance (to be described in the

next section). The "concerned" beams are of an absolute angle not more than $(\pi/2)$ from the robot orientation (3.8). *Nearest concerned range*, $R_{\min}(\vartheta)$, is defined as the shortest range value among the concerned beams. For the safety of navigation, the robot velocity is constantly adjusted based on the value of $R_{\min}(\vartheta)$. It will be set as relatively high when $\mathcal{R}$ is far from the "concerned" obstacles; and vice versa as follows:

$$v_t = \begin{cases} \frac{|P_t P_{\mathrm{IG}}|}{\Delta t}, & \text{if } \frac{|P_t P_{\mathrm{IG}}|}{\Delta t} < \frac{R_{\min}(\vartheta)}{R_{\min}(\vartheta)+R_{\mathrm{OB}}} v_{\mathrm{opt}} \\ \frac{R_{\min}(\vartheta)}{R_{\min}(\vartheta)+R_{\mathrm{OB}}} v_{\mathrm{opt}}, & \text{otherwise,} \end{cases} \tag{3.9}$$

where $v_{\mathrm{opt}} < v_{\max}$ denotes the reasonable speed that $\mathcal{R}$ can drive stably in an obstacle-free environment, and R_{OB} is the obstacle influence range given as Eq. (3.19).

In Eq. (3.9), by comparing the velocity magnitude with $(|P_t P_{\mathrm{IG}}|/\Delta t)$, the robot is ensured not to pass beyond the IG after driving at the obtained speed for a duration of Δt. In addition, even if $R_{\min}(\vartheta) \gg R_{\mathrm{OB}}$ or $R_{\min}(\vartheta) \ll R_{\mathrm{OB}}$, due to the robot's physical dimension and the limit of range measurements, v_t will be bounded by

$$\frac{R_{\mathrm{rob}}}{R_{\mathrm{rob}} + R_{\mathrm{OB}}} v_{\mathrm{opt}} \leq v_t \leq \frac{R_d}{R_d + R_{\mathrm{OB}}} v_{\mathrm{opt}}. \tag{3.10}$$

3.3.2 Search Range for Instant Goal Determination

The search range to find an IG is restricted to a special scope of the obstacle boundary, in order for the robot to move *forward* along the boundary of an obstacle in the desired direction, even in the presence of other obstacles. Three critical parameters to determine a search range are defined first: *SchFrom* and *SchEnd* are the beam indices from which the search process starts and ends, respectively, while *SchDir* is the left (CCW) or right (CW) hand side of the SchFrom direction. Then, *search range* of IG candidates (denoted as IG*s), given that the robot is currently at P_t, is expressed as follows:

$$\Theta = \partial\theta_{P_t}^{SchDir}(SchFrom, SchEnd). \tag{3.11}$$

Before searching for the k^{th} IG during following the i^{th} obstacle, the values of $\Theta_{i,k}$ and that of $SchFrom_{i,k}$ are required to be known ($SchEnd_{i,k}$ can be assigned with a value equal to $SchFrom_{i,k} + \pi$). Algorithm 2 describes how $SchFrom$ and Θ are determined. As shown in Fig. 3.6, *Actual Hit Point* $H_{i,k-1}^{\mathrm{Act}}, k = 1, 2, \cdots$, is defined as the obstacle point (on the boundary) nearest to $\mathcal{R}$ at the time when $G_{i,k}$ is to be determined. $H_{i,k-1}^{\mathrm{Act}}$ is obtained in the way as defined in Algorithm 2. $SchFrom_{i,k}$

is then set as the direction of $H_{i,k-1}^{\mathrm{Act}}$, and $\Theta_{i,k}$ is then given as in Eq. (3.11). The sequence of actions to obtain the search range can be summarized as follows:

$$G_{i,k-1} \quad \Rightarrow H_{i,k-1}^{\mathrm{Act}} \quad \Rightarrow SchFrom_{i,k} \quad \Rightarrow \Theta_{i,k}. \tag{3.12}$$

When $k = 1$, $G_{i,k-1}$ is undefined and we initialize $H_{i,k-1}^{\mathrm{Act}}$ (i.e. $H_{i,0}^{\mathrm{Act}}$) as H_i^1, the *near hit point* at the i^{th} *Start* (which is defined as the visible obstacle point on $\overrightarrow{S_iG}$). In other cases, the value of $G_{i,k-1}$ is required for computation of $H_{i,k-1}^{\mathrm{Act}}$.

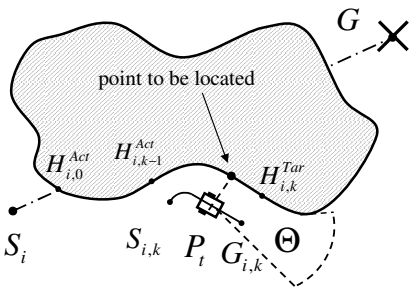


Figure 3.6: Find out point $H_{i,k}^{\mathrm{Act}}$ and thus $SchFrom_{i,k+1}$ and $\Theta_{i,k+1}$.

Algorithm 2 GetSearchRange($G_{i,k}$).

if $k = 0$ **then**
  $H_{i,k}^{\mathrm{Act}} \leftarrow H_i^1$ ▷ Initialize the first Actual Hit Point.
else
  $H_{i,k}^{\mathrm{Act}} := H_{i,k}^{\mathrm{Tar}} \leftarrow$ obstacle point nearest to $G_{i,k}$.
  if $\mathcal{R}$ is not at $G_{i,k}$ **then**
    $H_{i,k}^{\mathrm{Act}} \leftarrow$ obstacle point nearest to $\mathcal{R}$
    if $\overrightarrow{P_tH_{i,k}^{\mathrm{Act}}} \ni \partial\theta_{P_t}^{SchDir}(H_{i,k-1}^{\mathrm{Act}}, H_{i,k}^{\mathrm{Tar}})$ **then**
      $H_{i,k}^{\mathrm{Act}} \leftarrow H_{i,k-1}^{\mathrm{Act}}$ ▷ $H_{i,k}^{\mathrm{Act}}$ is ensured to be between $H_{i,k-1}^{\mathrm{Act}}$ and $H_{i,k}^{\mathrm{Tar}}$.
    end if
  end if
end if
$SchFrom_{i,k+1} \leftarrow$ direction of $H_{i,k}^{\mathrm{Act}}$; Obtain $\Theta_{i,k+1}$ using Eq. (3.11)

In the remaining part of this subsection, more descriptions are provided on how $H_{i,k}^{\mathrm{Act}}$ is determined according to three different cases. Note that no matter the current IG is reached or not, there may exist a situation where a new IG needs to be determined. For example, the current IG is not reachable, or approaching it any further will cause the robot to follow an obstacle that is not desired.

i) The IG has been reached, i.e. $|P_tG_{i,k}| \leq \varepsilon_{\mathrm{IG}}$ or $|P_tG_{i,k}| \ll |S_{i,k}G_{i,k}|$. Reasonably, take the current Target Hit point obtained as the Actual Hit point:

$$H_{i,k}^{\mathrm{Act}} = H_{i,k}^{\mathrm{Tar}}.$$

ii) Otherwise, it will be approximated by the point that $\mathcal{R}$ encounters first by imagining that its dimensions are big enough for it to contact the boundary. $H_{i,k}^{\text{Act}}$ is determined in a way that $\overrightarrow{P_t H_{i,k}^{\text{Act}}}$ is perpendicular to $\overrightarrow{S_{i,k}G_{i,k}}$ such that:

$$\mathbf{n}_{P_t H_{i,k}^{\text{Act}}} = \begin{bmatrix} \cos\theta_\perp & -\sin\theta_\perp \\ \sin\theta_\perp & \cos\theta_\perp \end{bmatrix} \mathbf{n}_{S_{i,k}G_{i,k}},$$

where $\theta_\perp$ is $(\pi/2)$ if $SchDir$ is $RIGHT$ or $-(\pi/2)$ otherwise.

iii) In case (ii), if $\overrightarrow{P_t H_{i,k}^{\text{Act}}}$ is not within $\partial\theta_{P_t}^{SchDir}(H_{i,k-1}^{\text{Act}}, H_{i,k}^{\text{Tar}})$. Then $\mathcal{R}$'s current position, P_t, may not lie exactly on line segment $\overrightarrow{S_{i,k}G_{i,k}}$. Furthermore, since $\mathcal{R}$ is required to follow the boundary forward in the direction of $SchDir$, point $H_{i,k}^{\text{Act}}$ should be "between" points $H_{i,k-1}^{\text{Act}}$ and $H_{i,k}^{\text{Tar}}$ on the boundary. Considering these two factors, in this case, $H_{i,k}^{\text{Act}}$ is set conservatively as $H_{i,k-1}^{\text{Act}}$, i.e.

$$H_{i,k}^{\text{Act}} = H_{i,k-1}^{\text{Act}}.$$

3.3.3 Algorithm to Determine Instant Goals

After the search range $\Theta = \partial\theta_{S_{i,k}}^{SchDir}(SchFrom, SchEnd)$ is determined, an IG will be obtained by checking each beam within it using the following procedure. Let variables j_{cur} and j_{old} record the current and last "searched" beams respectively, and initially set as $j_{\text{cur}} = j_{\text{old}} = SchFrom$.

Step 1: Set j_{cur} as the index of the next beam in the $SchDir$ within the search range. There will be an IG candidate in the direction of current or last "searched" beam if either i) the distance between the two successive measured points exceeds a certain threshold $\varepsilon_r > 2\sigma_{Rd}$, i.e.,

$$\|p_{j\text{cur}} - p_{j_{\text{old}}}\| = \sqrt{\hat{R}_{j\text{cur}}^2 + \hat{R}_{j_{\text{old}}}^2 - 2\hat{R}_{j\text{cur}}\hat{R}_{j_{\text{old}}}\cos\frac{2\pi}{N_s}} \geq \varepsilon_r, \tag{3.13}$$

or ii) after checking a certain number (for example five) of beams, Eq. (3.13) does not hold for any of them.

If there is an IG candidate, go to Step 2, otherwise, repeat Step 1.

Step 2: Determine the parameters of IG*:

The direction of IG* is set to be that of the longer of the ${j_{\text{cur}}}^{th}$ and ${j_{\text{old}}}^{th}$ beams:

$$j_{\text{IG}^*} = \begin{cases} j_{\text{cur}}, & \text{if } \hat{R}_{j\text{cur}} > \hat{R}_{j_{\text{old}}} \\ j_{\text{old}}, & \text{otherwise.} \end{cases} \tag{3.14}$$

If $\hat{R}_{j_{\mathrm{IG}^*}} \geq R_d - \sigma_{R_d}$, i.e. no obstacle is detected in the direction of IG*, $\mathcal{R}$ can move along the direction as far as $(\hat{R}_{j_{\mathrm{IG}^*}} - R_{\mathrm{rob}})$ (there is little knowledge of the environment beyond this distance). Otherwise, $\mathcal{R}$ can go as far as $(\hat{R}_{j_{\mathrm{IG}^*}} - 2R_{\mathrm{rob}})$ rather than $(\hat{R}_{j_{\mathrm{IG}^*}} - R_{\mathrm{rob}})$ so as not to collide with the obstacles nearby. That is, the distance of IG* is given by

$$r_{\mathrm{IG}^*} = \begin{cases} \hat{R}_{j_{\mathrm{IG}^*}} - R_{\mathrm{rob}}, \text{ if } \hat{R}_{j_{\mathrm{IG}^*}} \geq R_d - \sigma_{R_d}, \\ \max(\hat{R}_{j_{\mathrm{IG}^*}} - 2R_{\mathrm{rob}}, r_0), \text{ otherwise,} \end{cases} \tag{3.15}$$

where r_0 is a constant to ensure that r_{IG^*} is not shorter than a minimum value.

Step 3: IG* will be taken as IG in either of the following two cases:

i) The distance between two successive measured points $p_{j_{\mathrm{old}}}$ and $p_{j\mathrm{cur}}$ (as shown in Fig. 3.7) is greater than the robot size, i.e.,

$$\|p_{j\mathrm{cur}} - p_{j_{\mathrm{old}}}\| > 2R_{\mathrm{rob}}. \tag{3.16}$$

If Eq. (3.16) is satisfied, there is possibly a passage between the two points and thus check if an IG exists near them. As shown in Fig. 3.7, point M is on line segment $\overrightarrow{p_{j_{\mathrm{old}}} p_{j\mathrm{cur}}}$ and is of a distance $\min(\|p_{j\mathrm{cur}} - p_{j_{\mathrm{old}}}\|/2, R_{\mathrm{OB}})$ from point $p_{j\mathrm{cur}}$. Point C is on the same side as the robot with respect to the line $\overrightarrow{p_{j_{\mathrm{old}}} p_{j\mathrm{cur}}}$ and is perpendicular to the line with a distance of $(R_{\mathrm{OB}} + R_{\mathrm{rob}})/2$ from point M. Then the neighboring area NEIGH to choose IG*s is given by a disc $\mathcal{C}(C, (R_{\mathrm{OB}} - R_{\mathrm{rob}})/2)$. There will be a point in the area that $\mathcal{R}$ can reach safely if

$$\{j | \mathrm{SafeSP}(\gamma_{j,r_j}(R_{\mathrm{rob}} + \sigma_{R_d})) = \mathrm{True}, (r_j, j) \in \mathrm{NEIGH}, j \in \Theta\} \neq \emptyset. \tag{3.17}$$

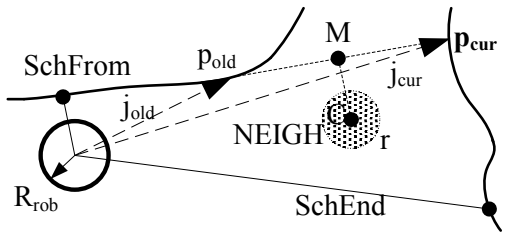


Figure 3.7: Determination of neighboring area NEIGH.

ii) There exists a safe path with length r_{IG^*} in the direction of the $j_{\mathrm{IG}^*}{}^{th}$ beam, that is,

$$\mathrm{SafeSP}(\gamma_{j_{\mathrm{IG}^*}, r_{\mathrm{IG}^*}}(R_{\mathrm{rob}} + \sigma_{R_d})) = \mathrm{True}. \tag{3.18}$$

In the first case, let j_{near} denote the nearest one in the set $\mathcal{J}$ with respect to the ${j_{\text{old}}}^{th}$ beam and set $j_{\text{IG}^*} := j_{near}$ and $r_{\text{IG}^*} := r_{j_{near}}$. In either case, an IG is found. If neither case is met, continue the IG search cycle.

Remark 3.2. *During following of the boundary of an obstruction $\mathcal{O}$, if there is an obstacle, say $\mathcal{O}_{\text{dis}}$, very close to obstacle $\mathcal{O}_i$ and the robot, the approach will determine the obstacle to follow in the following manner: if there is a passage (a safe path with length bigger than R_{rob}) between the two obstacles, an Instant Goal will be generated between the two obstacles and the robot will continue to follow obstacle $\mathcal{O}_i$; otherwise, obstacle $\mathcal{O}_{\text{dis}}$ is regarded as part of obstacle $\mathcal{O}_i$ and the robot continues to follow the "expanded" obstacle.*

A number of path planning approaches navigate a robot through local goals. For example, the subgoal method [21] chooses subgoals as the points at a certain distance ϵ w.r.t. their associated obstacle vertices. The setting of a certain distance ϵ is to ensure that the subgoal for an edge is not on another obstacle ("which must be at least a distance $\geq 2\epsilon$ from the edge"). To achieve the task of following an obstacle, the Instant Goal approach has several advantages over the subgoal method:

(i) It does not require obstacles to have a certain geometric properties. In contrast, the subgoal method assumes a point robot and a polygonal environment consisting of convex polygonal obstacles. It is therefore not directly applicable to the navigation of a physical robot in a real environment.

(ii) The subgoal method sets a certain distance to ensure that the subgoal for an edge is not on another obstacle. The Instant Goal approach uses the concept of "safe path" and "passage" and thus is more appropriate to distinguish the disturbing obstacles from the desired one.

(iii) Just setting a certain distance may be not able to guarantee that further subgoals will be generated, for example if the next edge is currently invisible. In contrast, IGs are able to be determined in any environment containing complex obstacles due to the following reasons: i) IGs are dynamically determined based on new sensory information, unlike subgoals which are all determined at one time; and ii) as in Step 1 of Sec 3.4, the condition to determine an IG* ensures that there is always an IG candidate that will be obtained within the searching scope.

3.3.4 Potential Field Method for Collision Avoidance

The generation of an IG has taken the locations of obstacles into account as

well as the robot dimensions and the range uncertainty (σ_{R_d}). Boundary following through IGs can solve obstacle avoidance problem to some extent. However, due to the unpredictability of the surrounding environment, sensor noise and imperfectness of control, the robot may still face the threat of collisions. In order to fully guarantee the safety of navigation, a special obstacle avoidance is considered here when some obstacles are near enough in the moving direction of the robot.

Critical Obstacle Avoidance

For a robot with velocity v_t in the duration of Δt, a collision with one or more obstacle(s) is possible only when it is within a certain range of the obstacles. If $\mathcal{R}$ is out of this range, it can move safely with the velocity in any direction. This *obstacle influence range*, denoted as R_{OB}, can be represented with respect to $\mathcal{R}$ as a circle centered at $\mathcal{R}$ with radius

$$R_{\text{OB}} = R_{\text{rob}} + v_t \Delta t + \sigma_{R_d}. \tag{3.19}$$

However, even if an obstacle is within this influence range, it will not directly affect the motion of $\mathcal{R}$ to an IG if the obstacle is *behind* the IG. This *IG influence range* is given by

$$R_{\text{IG}} = |P_t P_{\text{IG}}| + R_{\text{rob}} + \sigma_{Rd}. \tag{3.20}$$

Our strategy is that obstacle avoidance is triggered only when the nearest concerned range $R_{\min}(\vartheta)$ is within R_{OI}, the minimum value of R_{OB} and R_{IG}, i.e.

$$R_{\min}(\vartheta) \leq R_{\text{OI}} = \min(R_{\text{OB}}, R_{\text{IG}}). \tag{3.21}$$

At each time instant, every detected obstacle within the influence circle R_{OI} exerts on the robot a repulsive force which opposes the direction from the center of the robot to the obstacle. The total repulsive force exerted on the robot by all obstacles is the sum of the repulsive force in each beam direction. The repulsive potential U_j generated by the obstacle in the j^{th} beam direction is constructed in a way similar to [98]:

$$U_j = \begin{cases} \frac{1}{2}\left(\frac{1}{\hat{R}_j} - \frac{1}{R_{\text{OI}}}\right)^2, & \text{if } \hat{R}_j \leq R_{\text{OI}} \\ 0, & \text{if } \hat{R}_j > R_{\text{OI}}. \end{cases} \tag{3.22}$$

Only the concerned range data need to be considered for collision avoidance, and those of a smaller angle relative to the direction of the robot velocity affect motion

more significantly. To reflect these factors, a coefficient function is introduced:

$$\lambda(j) = \frac{1 + \mathrm{sgn}(\frac{\pi}{2} - \angle_{j,j_{\mathrm{IG}}})}{2}(C_0 + \frac{\frac{\pi}{2} - \angle_{j,j_{\mathrm{IG}}}}{\frac{\pi}{2}}), \tag{3.23}$$

where $\angle_{j,j_{\mathrm{IG}}} \in [0, \pi]$ denotes the absolute angle between the j^{th} and ${j_{\mathrm{IG}}}^{th}$ beam directions; the parameter C_0 is a positive constant, for instance 0.1, to make sure that the obstacles detected in the direction perpendicular to v_t may also have some effect (though relatively small) on collision avoidance.

With the information of both obstacle ranges and distance from IG, *obstacle influence map* $\tilde{\mathbf{R}}$ is used to reveal in which direction there exist obstacles that may collide with the robot:

$$\tilde{\mathbf{R}} = \left[\frac{1 - \mathrm{sgn}(\hat{R}_j - R_{\mathrm{OI}})}{2}\lambda(j)\right] \in \mathbb{R}^{\mathrm{N_s}}, \tag{3.24}$$

which implies that $\tilde{\mathbf{R}}$ will be nonzero only if $\hat{R}_j < R_{\mathrm{OI}}$.

The repulsive force required to avoid the obstacles in the direction of the j^{th} beam is then given by

$$\mathbf{f}_j = \tilde{R}_j \left(\frac{1}{R_{\mathrm{OI}}} - \frac{1}{\hat{R}_j}\right)\frac{1}{\hat{R}_j^2}, \tag{3.25}$$

where only for $\tilde{R}_j \neq 0$ does $\mathbf{f}_j$ need to be calculated.

Integration of Collision Avoidance into Boundary Following

During boundary following, collision checking is made at each time instant. Normally $\mathcal{R}$ moves directly toward an IG except when there are obstacles detected as near as in Eq. (3.21), in which case $\mathcal{R}$ will move in a different direction temporarily (for a period of Δt). There are two basic behaviors: motion toward the IG (called *Instant Goal driven* behavior or $\mathcal{IG}$ behavior), and motion due to the repulsive force generated by close obstacles (called *Obstacle Avoidance* behavior or $\mathcal{OA}$ behavior). Coordination of the two behaviors is via vector summation, which retains the advantage of coordination in the motor schema method [9, 10], i.e., simplicity. Obstacle avoidance is integrated into the whole navigation in a way without affecting the nature of boundary following. After that, if obstacle avoidance is no longer necessary, $\mathcal{R}$ will switch back to move directly toward an IG.

After the IG is determined, $\mathcal{IG}$ behavior can be expressed in the robot body coordinate frame by a unit vector originating from the center of the robot and pointing

to the IG:

$$\mathbf{F}_{\mathcal{IG}} = [\cos\theta_{\mathrm{IG}} \ \ \sin\theta_{\mathrm{IG}}]^T \in \mathbb{R}^2. \tag{3.26}$$

It can be known that the total repulsive force summed by Eq. (3.25) is inversely proportional in its magnitude to a cubic distance. A scalar $\xi > 0$ is thus used to scale the repulsive force to a level comparable to $\mathbf{F}_{\mathcal{IG}}$:

$$\mathbf{F}_{\mathcal{OA}} = \xi \sum_{j=1}^{N_s} \mathbf{f}_j. \tag{3.27}$$

Considering that, if the range value of the j_{IG} beam is as short as the minimum collision-free distance R_{rob}, the magnitude of the repulsive force in this beam direction should be much greater than $\mathbf{F}_{\mathcal{IG}}$, in order to make sure that $\mathcal{OA}$ behavior dominates in the behavior combination. In addition, it is noted that in this case $\lambda(j_{\mathrm{IG}}) = 1$ holds for Eqs. (3.23) and (3.24). In view of these, from Eqs. (3.27) and (3.25), ξ can be determined via the following equation:

$$\xi \left(\frac{1}{R_{\mathrm{rob}}} - \frac{1}{R_{\mathrm{OI}}} \right) \frac{1}{R_{\mathrm{rob}}^2} = C_f, \tag{3.28}$$

where C_f is a predefined constant much greater than 1.

The orientation of $\mathcal{R}$, initially set as in Eq. (3.8), is now re-determined by the summation of the vectors $\mathbf{F}_{\mathcal{IG}}$ and $\mathbf{F}_{\mathcal{OA}}$:

$$\vartheta = \angle(\mathbf{F}_{\mathcal{IG}} + \mathbf{F}_{\mathcal{OA}}). \tag{3.29}$$

3.4 Instant Goal Based Convergent Path Planning

Based on the Instant Goal approach of boundary following rather than assuming a perfect capability of boundary following, a realistic globally convergent path planner is presented in this section for mobile robot navigation.

3.4.1 Path Planner Design

Leave Condition

A Bug-like strategy is used to switch between the two basic motion modes. Initially, the robot moves directly toward the goal until it encounters an obstacle in its

direct path. At each *Start*, if no obstructing obstacle is detected, $\mathcal{R}$ moves directly toward the goal until one is detected; otherwise, it will enter boundary following mode and move along the boundary of the current blocking obstacle using the Instant Goal approach.

During boundary following, upon the receipt of a laser scan, if a certain condition is satisfied, $\mathcal{R}$ will leave from current blocking obstacle $\mathcal{O}_i$, and either move toward the goal or start following another obstacle. The robot's current position is denoted as "leave point" L_i, and in the meantime it will be set as the next *Start*, S_{i+1}.

In the work of [2], a basic *leave condition* is proposed as follows: $\mathcal{R}$ moves along the line segment $\overrightarrow{SG}$ until an obstacle, $\mathcal{O}_i$, is encountered where a hit point H_i^1 is defined; then it follows the boundary of the obstacle until $\overrightarrow{SG}$ is met at a distance d from G such that $d < |H_i^1 G|$, where a new hit point H_{i+1}^1 is defined. After leaving the obstacle, $\mathcal{R}$ continues to move along $\overrightarrow{SG}$. To improve navigation performance such as shorter paths, some Bug algorithms [17, 18, 20] adapt the tangent graph to obtain the locally shortest path, define several different transition conditions for switching between the two motion modes.

In this research, we propose the following *leave condition*, upon which the robot will leave the obstacle:

Case (i): no obstructing obstacle is detected and the distance from the goal is less than $|S_i^1 G|$; or

Case (ii): a new obstacle, rather the original obstructing one, is detected to be obstructing, and the distance from current near hit point H_{i+1}^1 to the goal is less than $|H_i^1 G|$.

Test of Goal Reachability

It is natural to think that the goal is unreachable if the robot has returned to a previous position after having traversed the whole boundary of an obstacle. But in the presence of sensing uncertainties, it is not easy to accurately determine whether a location has been accessed for a second time. Furthermore, returning to a previous position is only a special case that the goal is unreachable.

Lemma 3.1. *While following an obstacle boundary from Start S_i, if $\mathcal{R}$ traverses the line segment $\overrightarrow{H_i^1 H_i^2}$ in the same direction twice, then $\mathcal{R}$ completes a loop around the obstacle, which implies that the goal is unreachable.*

Proof: If the current moving mode is decided to be boundary following, subsequently $\mathcal{R}$ will attempt to follow the blocking obstacle continuously. When $\overrightarrow{H_i^1 H_i^2}$ is traversed by $\mathcal{R}$ in the same direction for a second time, it means that $\mathcal{R}$ has traveled at least 360^o around all the obstacle points on boundary ∂R_i (note that boundary ∂R_i may be inside the obstacle $\mathcal{O}_i$). This implies that either the robot or the goal is trapped – in either case, the goal is unreachable.

Lemma 3.1 relaxes the requirement that the robot need to return to the hit point H_i^1 before it identifies the next hit point. In addition, $\mathcal{R}$ is not required to return exactly to a previously covered position in order to test the goal reachability. The opposite situation is that the robot rather than the goal is trapped, where the goal is also not reachable.

Convergence Analysis

While $\mathcal{R}$ moves toward the goal, it may meet a series of obstacles, which are sequentially indexed. Suppose that i is the index of the current blocking obstacle. S_i denotes the i^{th} *Start* and accordingly H_i^1 denotes the near hit point at *Start* S_i, as shown in Fig. 3.8. An analysis about the convergence of the proposed path planner under the proposed leave condition is made as follows:

i) if $\mathcal{R}$ leaves the boundary according to Case (ii) of the proposed leave condition, it will ensure that

$$|H_i^1 G| > |H_{i+1}^1 G|. \tag{3.30}$$

For example, in Fig. 3.8, $|H_{i+1}^1 G| > |H_{i+2}^1 G|$ and $|H_{i+2}^1 G| > |H_{i+3}^1 G|$ hold.

ii) if $\mathcal{R}$ leaves the boundary according to Case (i) of the proposed leave condition, we then have

$$\begin{cases} |S_i G| & > |S_{i+1} G| \\ |S_i G| & = |S_i H_i^1| + |H_i^1 G| \\ |S_{i+1} G| & = |S_{i+1} H_{i+1}^1| + |H_{i+1}^1 G| \\ |S_i H_i^1| & \leq R_d < |S_{i+1} H_{i+1}^1| \end{cases} \Rightarrow |H_i^1 G| > |H_{i+1}^1 G|. \tag{3.31}$$

For example, in Fig. 3.8, $|H_{i+3}^1 G| > |H_{i+4}^1 G|$ holds.

iii) if current motion mode is *direct moving to the goal*, it is known that $|H_k^1 G| > |H_{k+1}^1 G|$, since $\mathcal{R}$ moves along $\overrightarrow{S_k G}$ before an obstacle is detected. For example, in Fig. 3.8, $|H_i^1 G| > |H_{i+1}^1 G|$ holds.

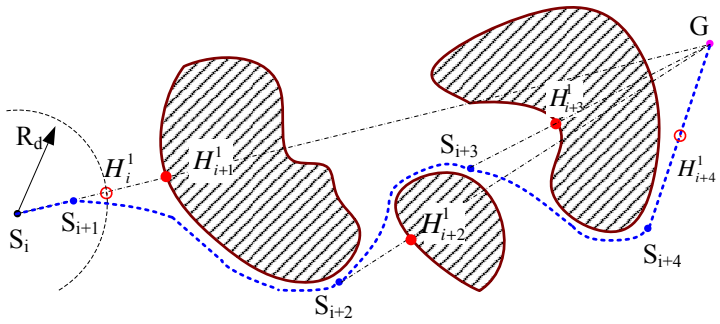


Figure 3.8: A path generated by the path planner with the proposed leave condition.

As the above has included all the three possible cases, it can be concluded that the distance between each near hit point and the goal decreases monotonically. Moreover, from the assumption that all obstacle boundaries are of a finite length, we know that the path length from a *Start* to the leave point is finite. If the goal is reachable, convergence to the goal is guaranteed by the *leave condition*. Let H^1_n denote the last hit point before the robot reaches the goal, the finite sequence of the distance function between the *Starts* and the goal can be given by

$$|H^1_1 G| > |H^1_2 G| > \cdots > |H^1_i G| > \cdots > |H^1_n G|. \tag{3.32}$$

3.4.2 Direction for Boundary Following

From the global performance standpoint, neither local direction for boundary following can be judged better than the other as long as no complete information is available. Local direction will directly affect the performance of subsequent boundary following. It can be deduced that the likelihood of obtaining a better global performance will increase by choosing a locally preferable direction at each *Start*. In the work of [18], the local direction is chosen based on the orientation of the boundary at the hit point (similar to H^1_i in this research), and it is expected that following the direction would take the robot closer to the goal. However, only partial information, that is, the information of the hit point, is utilized in that work.

In this research, determination of the following direction, *SchDir*, takes into account as much information of the blocking obstacle as possible. It is achieved by searching for an IG at the *Start* in a way similar to the "Algorithm to Determine Instant Goals" in Section 3.3.3, except that i) the search scope is now changed to the concerned beams; ii) checking is conducted simultaneously on the left- and right-hand sides of the $j_G{}^{th}$ beam direction; and iii) the search will be stopped only after all the

beams within the search scope have been checked.

Let $\mathscr{P}(j)$ denote the probability of being the IG direction for the j^{th} beam. When Eq. (3.13) is satisfied, $\mathscr{P}(j_{\mathrm{IG}^*})$ is given by

$$\mathscr{P}(j_{\mathrm{IG}^*}) = \min(1, \frac{\|p_{j_{\mathrm{cur}}} - p_{j_{\mathrm{old}}}\|}{2R_{\mathrm{rob}}}) + \frac{|\hat{R}_{j_{\mathrm{cur}}} - \hat{R}_{j_{\mathrm{old}}}|}{R_d} - \frac{\angle_{j_{\mathrm{IG}^*}, j_{\mathrm{G}}}}{\pi/2}, \tag{3.33}$$

where, on the right-hand side, the first term represents the possibility of a passage for $\mathcal{R}$ to go toward somewhere near the two successive obstacle points, the second term relates to how far the obstacles are in this direction, and the third term is about how big the angle is between this direction and the goal direction.

If in the meantime, Case (i) of Step 3 in Section 3.3.3 is satisfied, which means that there is a passage for $\mathcal{R}$ to go toward somewhere near the two successive obstacle points, the probability of the beam to be the final IG direction is greatly increased. Thus, a bonus value of 1 is added:

$$\mathscr{P}(j_{\mathrm{IG}^*}) = \mathscr{P}(j_{\mathrm{IG}^*}) + 1. \tag{3.34}$$

The maximum probability, $\mathscr{P}\mathrm{max}$, is updated with the obtained probability if it is greater than the recorded $\mathscr{P}\mathrm{max}$. The procedure then continues checking the next beams in the two directions.

Finally, the local direction for boundary following is determined according to the direction of the beam where $\mathscr{P}\mathrm{max}$ is found.

3.5 Simulation Studies

Extensive simulations were carried out to study the effect of the Instant Goal approach and the path planner. "Sensor data" are simulated with noise information based on the sensor's characteristics and behavior, such that the algorithms can be tested in a more realistic way.

3.5.1 Simulation Setup

In practice there are always some uncertainties associated with sensor readings. A *sensor model* describes how a sensor interacts with the environment and gives some information about the properties the sensor reports. For the time-of-flight range sensors like sonar or laser, the measured value represents reflection from the nearest surface (for sonar, specular reflection of the wave is not negligible). Since Moravec

and Elfes [33], a number of research interprets sonar sensor data in a probabilistic approach using a Gaussian distribution.

In the simulations, the probability density of the laser sensor is given by a constant $\mathscr{P}_c$ plus a Gaussian function which is multiplied by $\alpha(\rho)$, an attenuation of detection with distance:

$$\mathscr{P}(\rho|z,\theta) = \mathscr{P}_c + \frac{\alpha(\rho)}{2\pi\sigma_\rho\sigma_\theta} \exp\left[-\frac{1}{2}\left(\frac{(\rho-z)^2}{\sigma_\rho^2} + \frac{\theta^2}{\sigma_\theta^2}\right)\right], \tag{3.35}$$

where ρ is the range reading, θ is the angle with the optical axis of the sensor, z is the true space range value, variance σ_ρ^2 is a measure of range error, and variance σ_θ^2 is a measure of angular error. In Fig. 3.9, the vertical axis plots the likelihood of the detected object being at a given range and a given angle while the horizontal axes plot the distance from the range reading and the angle respectively.

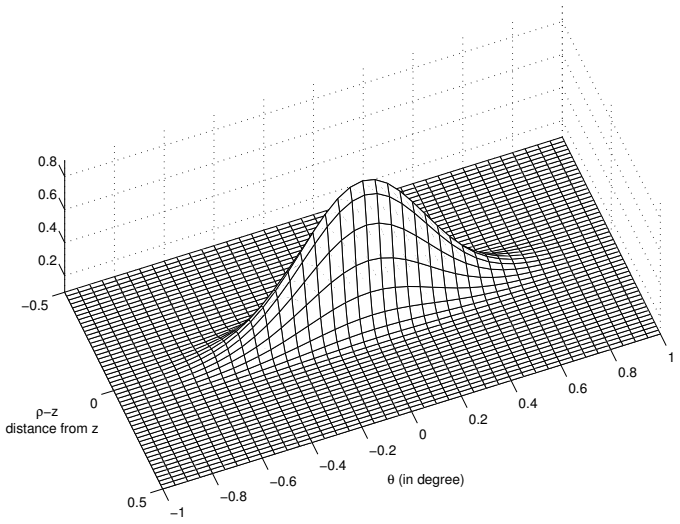


Figure 3.9: Sensor model of range readings of a laser rangefinder ($\sigma_\rho = 0.05$m, $\sigma_\theta =$ 0.25 degree, and $\mathscr{P}_c$=0.1).

In addition to the errors represented by range/angle variance, two other kinds of errors are introduced to the simulated perception of range values. Sometimes in some directions a laser rangefinder erroneously obtains maximum range measurements due to laser reflection of some objects, which may cause "holes" in the perception of the obstacles. Let $\mathscr{P}$randmax denote this probability for each direction. A laser rangefinder may also erroneously give some random measurements in some directions. This probability for each direction is denoted by $\mathscr{P}$randuni, and can be modeled as getting a random range reading with a uniform distribution between 0 and R_d.

In simulations, the j^{th} element of the vector representation (3.2) is obtained via Algorithm 3 in order to emulate[1] sensor errors of a laser rangefinder.

Algorithm 3 Add Sensor Error to the j^{th} Reading.

1: $\hat{\theta}_j \leftarrow \theta_j + \text{RandGau}(0, \sigma_\theta)$ ▷ Add angular error to θ_j
2: Obtain $\rho_{\hat{\theta}_j}$, the distance of obstacles in the direction of $\hat{\theta}_j$
3: **if** $\text{RandUni}(0, 1.0) < \mathscr{P}\text{randmax}$ **then**
4: $\quad \hat{R}_j \leftarrow R_d$
5: **else if** $\text{RandUni}(0, 1.0) < \mathscr{P}\text{randuni}$ **then**
6: $\quad \hat{R}_j \leftarrow \text{RandUni}(0, R_d)$
7: **else**
8: $\quad \hat{R}_j \leftarrow \rho_{\hat{\theta}_j} + \text{RandGau}(0, \sigma_r)$
9: **end if**

The basic settings of the simulation tests are listed as follows:

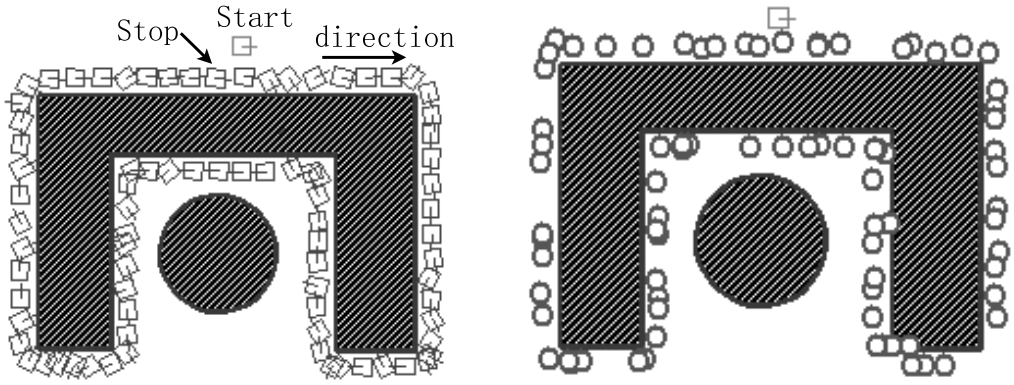
- An omni-directional robot is used with optimal velocity $v_{\text{opt}} = 0.40$ m/s.
- Laser rangefinders' detectable range is 15 m and its angular resolution is $\Delta\alpha_s = 1^\circ$ ($N_s = 360$).
- Period between two successive motions is set as $\Delta t = 0.5$ s and the robot trajectories are plotted at an interval of 10 periods.
- Parameters of sensor errors are $\sigma_\rho = 0.05$ m, $\sigma_\theta = 0.25^\circ$, $\mathscr{P}$randmax=0.01 and $\mathscr{P}$randuni=0.01.
- ε_r in Eq. (3.13), is set as 0.12 m.
- Obstacles can be configured to be of any shape and no model of the world was pre-created. Decisions were based directly on simulated range data obtained upon receipt of a laser scan.

3.5.2 Boundary Following

In the simulations, the effective radius of the robot is set as $R_{\text{rob}} = 0.40$m. The algorithm was first tested in an environment as shown in Fig. 3.10. The navigation began at the point labeled as "Start" in Fig. 3.10(a). The robot followed the obstacle for one full loop in a clockwise direction (labeled as a line with an arrow) and stopped

[1]The uniform random function $\text{RandUni}(\min, \max)$ generates a set of random variables x with a uniform distribution within the scope $[\min, \max]$. As opposed to a Gaussian function $\mathcal{N}(\bar{x}, \sigma)$, the Gaussian random function $\text{RandGau}(\bar{x}, \sigma)$ generates a set of random variables x with a Gaussian (or Normal) distribution.

at the location labeled as "Stop". The final robot trajectories are plotted with a series of rectangles where the orientation of the robot is denoted by a straight line on each rectangle. It is shown that the robot is able to track around both convex and concave corners even with obstacles nearby. Note that the trajectories are not completely parallel to the straight lines of the obstacle because the robot does not use any reasoning about the obstacle shape it follows.

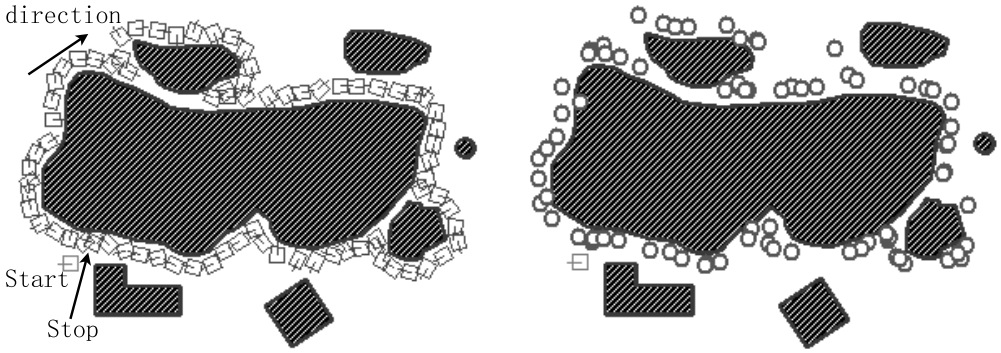


(a) Trajectories. (b) Instant Goals.

Figure 3.10: Following a large U-shaped obstacle consisting of convex and concave corners. A round obstacle is placed near the U-shaped obstacle.

The algorithm was then tested in a densely cluttered environment as shown in Fig. 3.11. Initially, the robot was at the left bottom of the biggest obstacle in the figure. It then followed the obstacle for one full loop in a clockwise direction. It can be observed that the robot was able to track around the obstacle properly even with obstacles nearby. When the distance between two adjacent obstacle points is smaller than the effective size of the robot, it is reasonable to regard them as two points of the same obstacle even if they actually belong to two different obstacles. In addition, when there is a passage between two obstacles, the robot should continue following the original obstacle rather than the other one that had been detected.

The IGs during the robot's motion in the two environments are represented as a series of small circles in Fig. 3.10(b) and 3.11(b) respectively. It is shown that, generally, the generated IGs were able to guide the robot move forward correctly. The IGs are frequently generated or revised where they are densely plotted; sometimes they are located very close to or even within the obstacles. This will not affect the correct motion of the robot, since collision checking was made at each time instant and the IGs can be revised when they are found to be inappropriate.

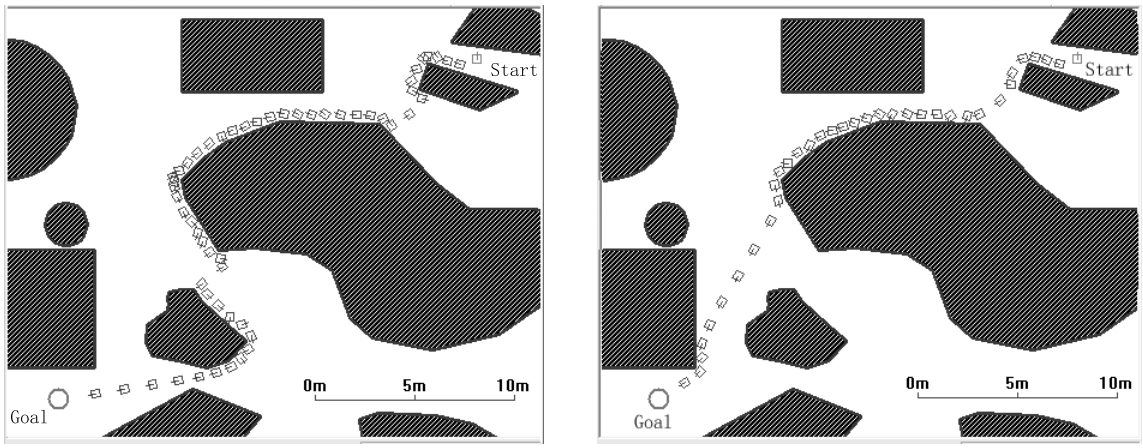


(a) Robot trajectories. (b) Instant Goals.

Figure 3.11: Following a complex curve with some disturbing obstacles.

3.5.3 Path Planner

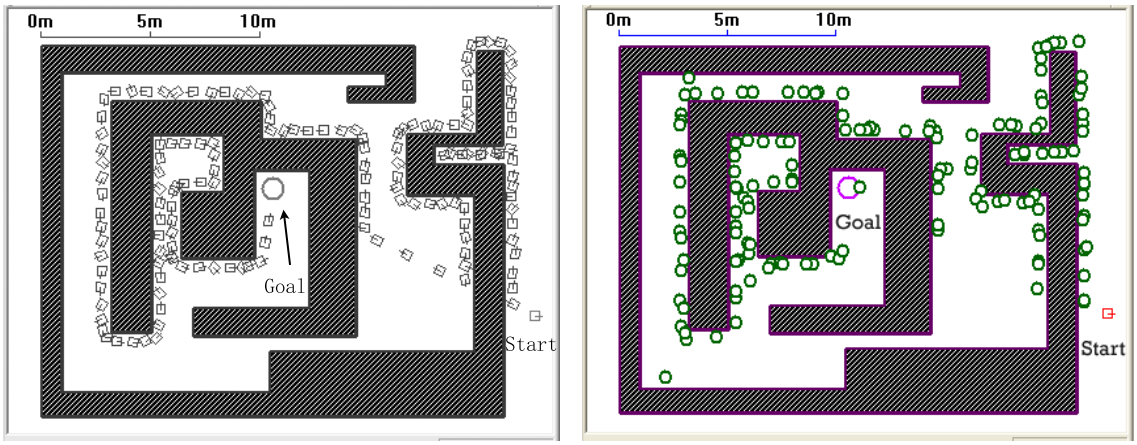
The effective radius of $\mathcal{R}$ is set as $R_{\text{rob}} = 0.30\text{m}$. The algorithm was tested in an environment with convex and concave obstacles. Figs. 3.12(a) and 3.12(b) show the robot trajectories under the basic leave condition and the proposed one, respectively. The local direction was determined using the same approach for searching IGs. The performance of a path planner can be evaluated by the path length and the time used. One can see that the proposed leave condition generated shorter paths (also with less time), which is generally the case for an environment of low obstacle density. However, this is not necessarily true in an obstacle-cluttered environment, in which case there are much less chances for $\mathcal{R}$ to detect a new obstructing obstacle rather than the original one before meeting the line $\overrightarrow{H_i^1 G}$.



(a) Basic leave condition. (b) Proposed leave condition.

Figure 3.12: Robot trajectories in a low obstacle density environment.

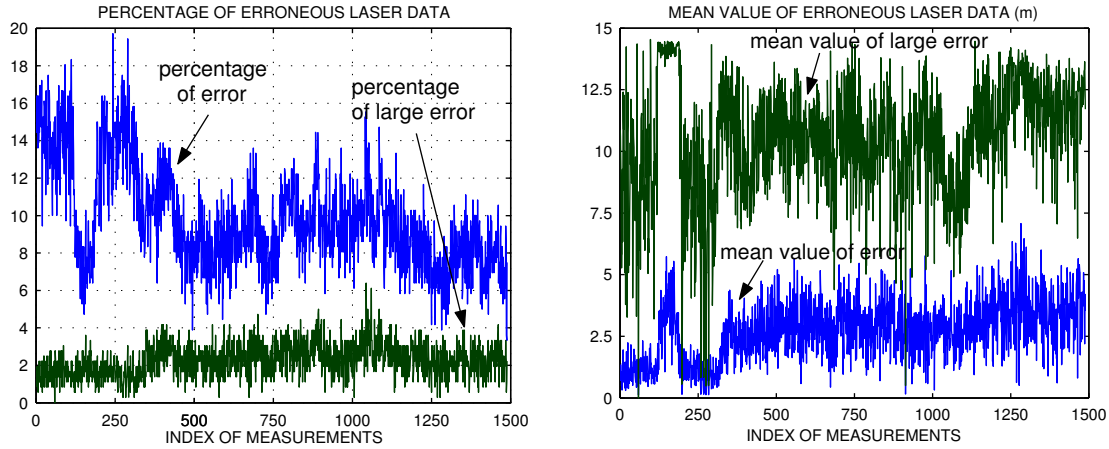
Figs. 3.13 (a) and (b) show the robot trajectories and the generated IGs obtained by the test in an obstacle-cluttered environment. Fig. 3.14 shows the statistics of the errors introduced to the sensor data in this simulation test. A "measurement" is regarded as "erroneous" if the distance between the measured point detected in some direction and the true obstacle point is greater than 0.10 m ($2\sigma_\rho$). It will be regarded as "large error" if the distance is more than 0.30 m ($6\sigma_\rho$). As shown in Fig. 3.14(a), in each scan of 360 "measured" range values, the percentage of erroneous measurements and "large error" are around 9% and 2.5%, respectively. Fig. 3.14(b) shows that the average value of the erroneous measurements and the "large error" are around 2m and 10m, respectively, under the setting of maximum range 15m. Other simulations were also carried out using laser data carrying noise at a similar level.



(a) Trajectories. (b) Instant Goals.

Figure 3.13: Navigation in a high obstacle density environment.

All the tests are based on range data from the laser rangefinder, and there is no requirement for an analytical expression of obstacle boundaries. It is shown that during boundary following, the robot can detect a small gap, and then enter into it to check if there is a passage there, which is important for the robot not to miss any passage during boundary following. It is also shown that the Instant Goal approach is suitable for discrete, noisy range data as the generation of IGs does not solely rely on a one time scan, and a wrong decision is able to be corrected in subsequent steps. However, continuous presence of misleading and noisy data with large errors may cause the approach to fail, e.g., moving back along the obstacle or, even worse, following a diverging path.



(a) Percentage of errors.

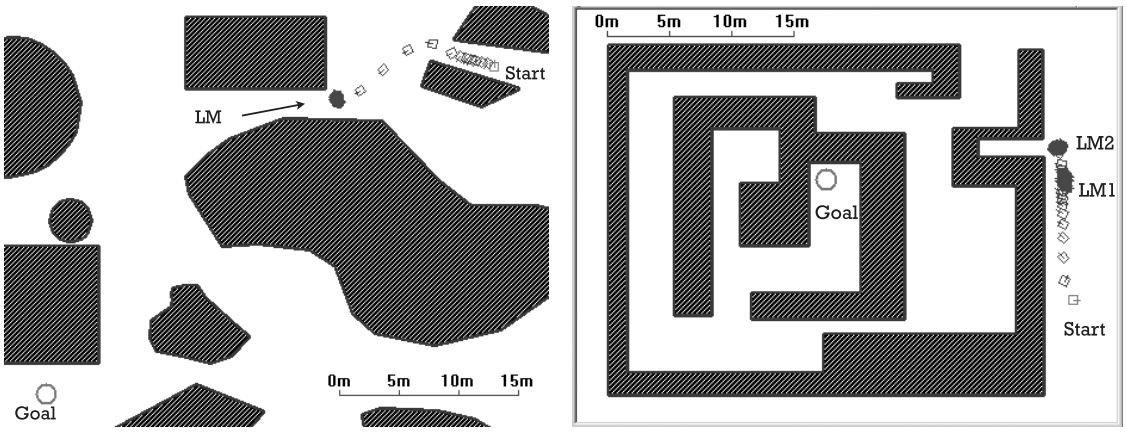
(b) Mean of erroneous values.

Figure 3.14: Statistics of erroneous simulated measurements. The horizontal axes plot the index of laser scans. The vertical axes plot "percentage of errors" and "mean of erroneous values" in diagrams (a) and (b), respectively.

3.5.4 Comparison with Other Approaches

Sensor-based local path planners such as potential field methods and behavior-based systems use local sensory information in a purely reactive fashion, and thus are usually much simpler to implement than global ones. However, they may get trapped in a local minimum, and subsequently follow a diverging path or a loop while attempting to escape from the local minimum. Fig. 3.15 shows the results of navigation of the mobile robot using a classical potential field method, in the same environments as in Figs. 3.12 and 3.13. Though the algorithm itself is much simpler than the Instant Goal approach, the robot may be easily trapped at local minimums. In Fig. 3.15(a), the robot was stuck around the location $LM1$. As shown in Fig. 3.15(b), at first, the robot was stuck around the location $LM1$ for some time, and then got trapped at the location $LM2$ without being able to move out of the local minimum. In the simulations, sensor errors were added at a level similar to before, which however was unable to help the robot to go out from local minima.

Like the Bug algorithms, the Instant Goal approach combines global information and local sensory data, and is able to achieve convergence to the goal. In addition, it does not need to maintain a global map of the overall navigation environment as required by global sensor-based path planners. Therefore, it shares the property of real-time planning like reactive systems. Compared with the Bug algorithms, the Instant Goal approach is more practical for a real world application considering the



(a) Low obstacle density. (b) High obstacle density.

Figure 3.15: Results of navigation using a classic potential field method.

following aspects:

(i) It does not assume a perfect capability of boundary following, and the Instant Goal approach is used instead: by reaching a series of IGs, the robot moves "forward" along the boundary even if the obstacle is of arbitrary shape and "disturbing" obstacles are present.

(ii) The proposed approach is fully based on realistic (discrete, noisy) range data received from a rangefinder. In contrast, the range sensor is assumed to be able to present a disc of radius r_v [2, 16] or to provide perfect readings of the obstacles [17, 18].

(iii) Some algorithms [2,21] assume and represent the robot as a point, which results in the need to extract boundary expression from range information, and transform a physical robot into a point using the $\mathcal{C}$-space approach. In comparison, the proposed approach does not require a treatment of the robot as a point. By following the boundary of the obstacle for a non-point robot, the Bug strategy is applicable for it to achieve convergence to the goal.

(iv) The subgoal method [21] requires the environment to consist of only convex polygonal obstacles, which is obviously not applicable in a real environment. In comparison, the proposed approach makes no assumption about the geometric properties of obstacles.

A complete path planner can always guarantee global convergence. Bug algorithms [2, 17] are proved mathematically to be able to guarantee global convergence. Using

a Bug-like strategy to switch between motion modes, the proposed path planner is theoretically complete. However, global convergence may be violated if the robot is unable to leave the obstacle at the place as decided by the leave condition or boundary following is not achieved as expected. Such cases may be caused by the discrete, noisy nature of range sensors, other uncertainties (e.g. missing the detection of passing the line SG), or failure of planning a suitable motion (especially when the robot dynamics is taking into account).

3.6 Summary

This chapter presents a vector to represent the local environment, which saves much memory space and is suitable for behavior generation and obstacle avoidance. Then, we propose the Instant Goal methodology to keep the robot moving forward along the boundary of an obstacle of arbitrary shape even in the presence of disturbing obstacles. Based on this approach of boundary following, a practical globally convergent path planner is presented for mobile robot navigation in a unstructured, complex environment. The effectiveness of the algorithms has been validated through realistic simulations with noisy range data.

The main contributions of the research presented in this chapter are: i) an Instant Goal approach is proposed for collision-free boundary following along an obstacle of arbitrary shape. The robot is able to continuously follow the desired obstacle in the desired direction even in the presence of disturbing obstacles; ii) based on the Instant Goal approach, rather than assuming the capability of boundary following, a practical globally convergent path planner is presented for mobile robot navigation in unknown environments; and iii) the proposed approach does not require a range sensor to have perfect sensing and is based on discrete, noisy range sensory data.

Chapter 4

PPC Based Constrained Path Generation and Hybrid Dynamic Path Planning Approach

Complex curve is a direct method to solve the problem of planning smooth paths for nonholonomic mobile robots, since it is able to address dynamic and curvature constraints simultaneously. In this chapter, we investigate the use of polar polynomial curve (PPC) since it provides a closed-form expression for the robot position, which is convenient for robot path planning. Another motivation is that collision test, generally difficult for a complex curve, can be done efficiently using PPC curve's properties. Then, a hybrid path planning approach is presented in this chapter to guide the robot to follow an obstacle of arbitrary shape, by generating a proper "Instant Goal" (and a series of deliberate, feasible motions) and planning reactively when needed using a fuzzy controller for wall following. PPC curve based method is also applied for real-time path planning to produce smooth, feasible paths with continuous and upper-bounded curvature for car-like robots.

4.1 PPC Curve Based Smooth and Feasible Path Generation

This section presents a polar polynomial method to provide a smooth transition curve with a velocity profile satisfying the robot dynamic constraints.

4.1.1 PPC Curve

PPC Curve Connecting Lines

This subsection recalls the works in [78,79] which uses a polar polynomial curve to replace the circular arc which connects two line segments tangentially. Let Φ be the angle between the two line segments, and Ω be the circular center of the arc (denoted by dashed line) determined by the two lines. As shown in Fig. 4.1, a polar frame is established such that Ω is the origin and line $\overrightarrow{\Omega(\mathbf{q}_0}$ is the polar axis.

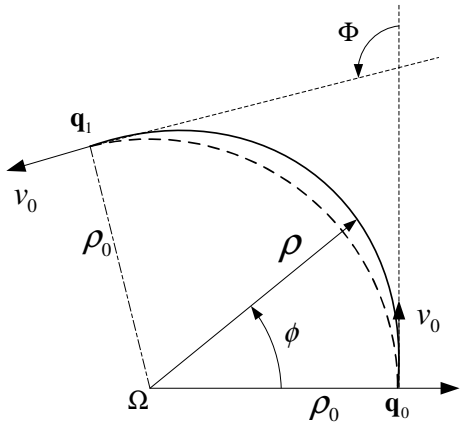


Figure 4.1: Polar polynomial curve connecting two straight lines.

The curve starts from $(\varrho_0, 0)$ and ends at (ϱ_0, Φ), where ϱ_0 is called the initial and/or final radii. The six constraints of the position, tangent (orientation) and curvature are given by:

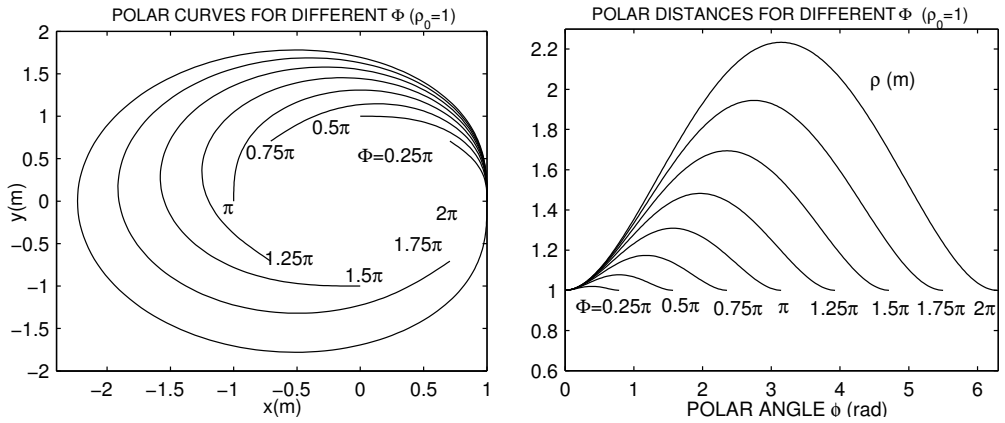
$$\begin{cases} \varrho = \varrho_0, \varrho' = 0, \kappa = 0, \text{ at } \phi = 0 \\ \varrho = \varrho_0, \varrho' = 0, \kappa = 0, \text{ at } \phi = \Phi, \end{cases} \tag{4.1}$$

which, together with Eqs. (2.7) and (2.11), yield a fifth order polar polynomial:

$$\varrho(\phi) = \varrho_0 \left(1 + \frac{\phi^2}{2} - \frac{\phi^3}{\Phi} + \frac{\phi^4}{2\Phi^2}\right) = \varrho_0 \left(1 + \frac{(\Phi - \phi)^2 \phi^2}{2\Phi^2}\right). \tag{4.2}$$

Fig. 4.2(a) shows polar polynomial curves designed as Eq. (4.2) (or called PPC for short) for different values of Φ. According to Eq. (4.2), the maximum range value of a PPC curve is $\varrho_0(1+\Phi^2/32)$ and occurs at $\phi = \Phi/2$. The function (not the curve) does not have a single concavity w.r.t. ϕ, as shown in Fig. 4.2(b).

A velocity profile can be associated with a PPC curve for it to be used as robot trajectory. As required by the boundary condition, the final speed is equal to the



(a) Shape of PPC curves. (b) ϱ projected to polar angle ϕ.

Figure 4.2: PPC curves for different Φ, with ϱ_0 set to be 1.

initial one, v_0. The velocity profile is designed such that $d\phi/dt$ is a constant, i.e.

$$\phi(t) = \frac{\Phi}{\tau}t, \tag{4.3}$$

where the duration of the motion τ can be obtained as follows by substituting the derivative of (4.3) and t=0 into Eq. (2.12):

$$\tau = \frac{\varrho_0 \Phi}{v_0}. \tag{4.4}$$

Properties of PPC Curves

Table 4.1 lists the important features of the function $\varrho(\phi)$ (4.2), where $\phi_1 = \frac{3-\sqrt{3}}{6}\Phi$ and $\phi_2 = \frac{3+\sqrt{3}}{6}\Phi$. As there are two inflexions (located at $\phi = \phi_1, \phi_2$), the function (not the curve) does not have a single concavity as shown in Fig. 4.2(b).

Table 4.1: Important Features of a PPC curve.

ϕ	$(0, \phi_1)$	ϕ_1	$(\phi_1, \frac{\Phi}{2})$	$\frac{\Phi}{2}$	$(\frac{\Phi}{2}, \phi_2)$	ϕ_2	(ϕ_2, Φ)
ϱ'	+	+	+	0	-	-	-
ϱ''	+	0	-	-	-	0	+

Property 4.1. *A PPC curve is symmetrical w.r.t. the line $\phi = \Phi/2$.*

Proof. $\forall\ (\Phi - \phi), \phi\ \in\ [0, \Phi]$, we have:
$(\Phi - \phi)\ \in\ [0, \Phi]$, and

$$\begin{aligned}\varrho(\Phi-\phi) &= \varrho_0\left(1+\tfrac{(\Phi-\phi)^2}{2}-\tfrac{(\Phi-\phi)^3}{\Phi}+\tfrac{(\Phi-\phi)^4}{2\Phi^2}\right)\\ &= \varrho_0\left(1+\tfrac{(\Phi-\phi)^2\phi^2}{2\Phi^2}\right) = \varrho(\phi) \text{ (according to Eq. (4.2)).}\end{aligned}$$ □

Property 4.2. *The function $\varrho(\phi)$ increases and decreases monotonically in the first and last half segments of the PPC curve, respectively.*

Proof. The first, second and third derivatives of the curve to the polar angle are:

$$\begin{cases}\varrho' = \varrho_0(\phi-\frac{3\phi^2}{\Phi}+\frac{2\phi^3}{\Phi^2}) = \frac{2\varrho_0}{\Phi^2}\phi(\phi-\frac{\Phi}{2})(\phi-\Phi)\\ \varrho'' = \varrho_0(1-\frac{6\phi}{\Phi}+\frac{6\phi^2}{\Phi^2}) \ = \varrho_0[\frac{6(\phi-\Phi/2)^2}{\Phi^2}-\frac{1}{2}]\\ \varrho''' = \frac{d^3\varrho}{d\phi^3} = \varrho_0(-\frac{6}{\Phi}+\frac{12\phi}{\Phi^2}).\end{cases} \tag{4.5}$$

According to Eq. (4.5),

$$\begin{cases}\varrho' > 0, \forall\ \phi\ \in\ (0,\Phi/2)\\ \varrho' = 0, \forall\ \phi = \Phi/2\\ \varrho' < 0, \forall\ \phi\ \in\ (\Phi/2,\Phi).\end{cases} \tag{4.6}$$

Therefore, the curve is monotonous in the first and second half segments. □

Property 4.3. *The curve combined by a PPC curve and two line segments is continuous in curvature.*

At $\phi = 0, \Phi$, both the PPC curve and the two line segments have a curvature of zero. In addition, the curvature of the PPC curve is continuous within the whole spectrum $[0, \Phi]$, if we substitute ϱ, ϱ' and ϱ'' (which are all continuous as in Eq. (4.5)) into the curvature (2.11).

Maximum Curvature of PPC Curve and Maximum Associated Velocity

To compute the maximum value of $v(t)$ in Eq. (2.12), we first consider that of $Z = \varrho^2 + \varrho'^2$. We have

$$\frac{dZ}{d\phi} = 4\frac{\varrho_0^2}{\Phi^2}\phi(\phi-\frac{\Phi}{2})(\phi-\Phi)[\frac{\phi^2}{2}(\frac{1}{\Phi}\phi-1)^2+\frac{6}{\Phi^2}(\phi-\frac{\Phi}{2})^2+\frac{1}{2}]. \tag{4.7}$$

The continuous function Z has extremum values at the roots ($\phi = 0, \Phi/2, \Phi$) of the equation $\frac{dZ}{d\phi} = 0$ and the boundaries ($\phi = 0$ and $\phi = \Phi$). The maximum value of Z (and thus that of $v(t)$) is found to be at $\phi = \Phi/2$. Thus, we have

$$\max(v(t)) = v|_{\phi=\frac{\Phi}{2}} = \varrho_0\left(1+\frac{\Phi^2}{32}\right)\frac{\Phi}{\tau} = \left(1+\frac{\Phi^2}{32}\right)v_0. \tag{4.8}$$

Fig. 4.3 plots the curvatures of PPC curves of different values of Φ for $\varrho_0 = 1$. For a PPC curve, the derivative of its curvature to ϕ is given by

$$\frac{d\kappa}{d\phi} = (-\varrho^3\varrho' - \varrho^3\varrho''' + 3\varrho^2\varrho'\varrho'' - 4\varrho\varrho'^3 + 3\varrho\varrho'\varrho''^2 - \varrho\varrho'^2\varrho''' - 3\varrho'^3\varrho'')/(\varrho^2 + \varrho'^2)^{\frac{5}{2}}. \tag{4.9}$$

As it can be verified in Eq. (4.9) that $\frac{d\kappa}{d\phi}|_{\phi=\frac{\Phi}{2}} = 0$ holds, one extremum value of the curvature occurs at $\phi = \Phi/2$ and its value is

$$\kappa|_{\phi=\frac{\Phi}{2}} = \frac{32(48+\Phi^2)}{\varrho_0(32+\Phi^2)^2}. \tag{4.10}$$

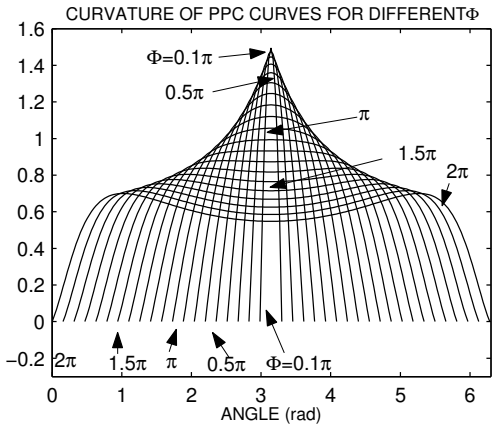


Figure 4.3: Curvatures of PPC curves ($\varrho_0 = 1$) for different Φ. The curvature of each curve reaches its maximum value(s) one time for small Φ, and twice for large Φ.

As shown in Fig. 4.4, the polar angle at which the maximum curvature occurs, $\phi_{\max}$, is $\Phi/2$, for any choice of ϱ_0 when $\Phi \leq 1.20\pi$. For $\Phi > 1.20\pi$, the ratio $\kappa_{\max}/\kappa|_{\phi=\frac{\Phi}{2}}$[1] can be well fitted by the following 3rd order polynomial (with the absolute error $\leq 0.001m^{-1}$):

$$f(\Phi) = -0.0054\Phi^3 + 0.1145\Phi^2 - 0.6221\Phi + 2.0063. \tag{4.11}$$

Therefore, the maximum value of the curvature of a PPC curve can be given as

$$\max(\kappa(\phi)) = \begin{cases} \kappa|_{\phi=\frac{\Phi}{2}}, & \text{if } \Phi \leq 1.20\pi \\ f(\Phi)\kappa|_{\phi=\frac{\Phi}{2}}, & \text{otherwise.} \end{cases} \tag{4.12}$$

[1] $\phi_{\max}$ is chosen to be the one not larger than $\Phi/2$ if there are two maximum curvatures.

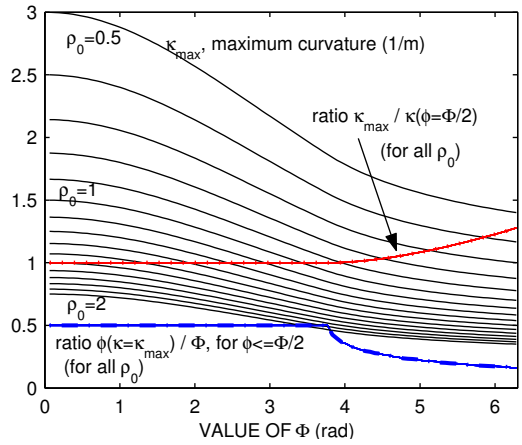


Figure 4.4: Maximum Curvature of PPC curves for different values of ϱ_0 are plotted vs. Φ. The bottom thick line plots the ratio $\phi_{\max}/\Phi$. The upper thick line plots the ratio $\kappa_{\max}/\kappa|_{\phi=\frac{\Phi}{2}}$.

Design Constraints Imposed by Curvature and Velocity

As the value in the square root in (2.41) cannot be negative and the maximum velocity of the curve is (4.8), we have

$$\kappa \leq \frac{\mu g}{[\max(v(t))]^2} = \frac{\mu g}{(1+\Phi^2/32)^2 v_0^2}. \tag{4.13}$$

Remark 4.1. *The work in [79] reported that the maximum curvature of PPC curve should be less than robot's maximum admissible centrifugal acceleration divided by square of robot velocity, which is obvious based on $a_n = \kappa v^2$. In comparison, our work directly relates the curvature limit to the friction constraint, considering that the limit of centrifugal acceleration is not specified for a robot.*

Furthermore, our work provides a way (see Section 4.1.1) to explicitly compute the maximum curvature. Based on it, we develop design constraints imposed by curvature and velocity in this subsection.

The velocity limit (2.42) and the maximum velocity (4.8) along a PPC curve impose the following constraint under the velocity profile design (4.3):

$$\varrho_0 \left(1 + \frac{\Phi^2}{32}\right) \frac{\Phi}{\tau} \leq v_{\max}. \tag{4.14}$$

The curvature constraints (2.37) and (4.13) impose the following constraint:

$$\max(\kappa) \leq \min\left(\frac{\mu g}{(1+\Phi^2/32)^2 v_0^2}, \frac{1}{r_{\min}}\right). \tag{4.15}$$

Deriving from Eqs. (2.33), (2.12) and (2.11), we have:

$$\omega(t) = v(t)\kappa(t) = \left(1 + \frac{\varrho'^2 - \varrho\varrho''}{\varrho^2 + \varrho'^2}\right)\frac{v_0}{\varrho_0}. \tag{4.16}$$

It can be known that $\left(1 + \frac{\varrho'^2 - \varrho\varrho''}{\varrho^2 + \varrho'^2}\right)$ is a function of ϕ and Φ without ϱ_0. By trying different values of Φ, its maximum value is found to occur at $\phi = \Phi/2$. Therefore, the maximum angular velocity equals to $(vk)|_{\phi=\frac{\Phi}{2}}$[2]. As the actuator limit (2.43) holds, we have the following constraint:

$$\max(\omega) = (vk)|_{\phi=\frac{\Phi}{2}} = \left(1 + \frac{16}{32 + \Phi^2}\right)\frac{v_0}{\varrho_0} \leq \omega_{\max}. \tag{4.17}$$

Differentiate Eq. (2.33), i.e. $\kappa = \tan\varphi/L$ to t, we have

$$\dot{\kappa} = \frac{d\kappa}{dt} = \frac{d\kappa}{d\phi}\frac{d\phi}{dt} = \frac{1}{L\cos^2\varphi}\frac{d\varphi}{dt} = \frac{\sqrt{1+\kappa^2}}{L}\dot{\varphi}. \tag{4.18}$$

Substitute Eqs. (4.4) and (4.3) into Eq. (4.18) and notice that Eq. (2.44) holds, we have

$$|\dot{\varphi}| = \frac{Lv_0}{\varrho_0}\frac{1}{\sqrt{1+\kappa^2}}\left|\frac{d\kappa}{d\phi}\right| \leq \dot{\varphi}_{\max}, \tag{4.19}$$

where $\frac{d\kappa}{d\phi}$ is given by Eq. (4.9). As shown in Fig. 4.5, the maximum value of $\frac{1}{\sqrt{1+\kappa^2}}\left|\frac{d\kappa}{d\phi}\right|$ (a function of (ϕ, φ)) vs. Φ can be approximated as a polynomial about Φ, say $f_{dk}(\Phi) = \max \frac{1}{\sqrt{1+\kappa(\Phi)^2}}\left|\frac{d\kappa(\Phi)}{d\phi}\right|$.

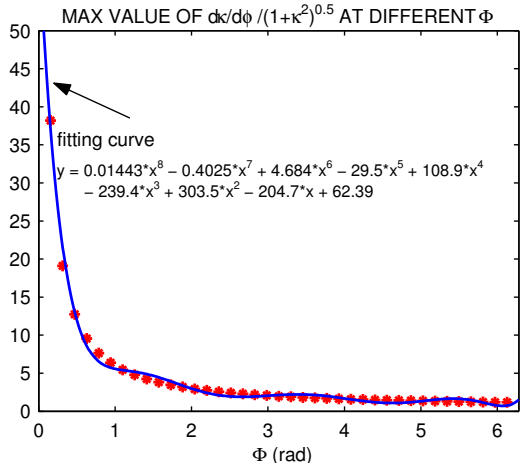


Figure 4.5: Maximum values of $\frac{1}{\sqrt{1+\kappa^2}}\left|\frac{d\kappa}{d\phi}\right|$ at different Φ.

Considering that Lv_0 is known and, for a particular ϱ_0 designed, the value of Φ can be determined geometrically, we are able to check whether the constraint (4.19) is satisfied. It is also noted that $f_{dk}(\Phi)$ is relatively small for $\Phi \geq 1$.

[2]This is reasonable due to the fact that $\max(v(t))$ occurs at $\phi = \Phi/2$ (Eq. (4.8)), and $\max(\kappa(t))$ occurs at $\phi = \Phi/2$ for $\Phi \leq 1.2\pi$.

4.1.2 Combination of Curves to Connect Two Configurations

Sensor-based path planning methods often navigate a robot from its current pose to its local destination without placing a requirement on the robot's final orientation. This section explores the use of a combination of curves to connect two configurations of this type.

PPC Curve for Robot Performing Translation

If the robot is currently performing a pure translation, we can use a PPC curve followed by a straight line segment to connect two configurations $\mathbf{q}_0$ and $\mathbf{q}_1$. In other words, a PPC curve is used to connect two straight line segments, which satisfies the required boundary conditions. As illustrated in Fig. 4.6, two circles of radius ϱ_0 are established such that they are tangential to the orientation of $\mathbf{q}_0$. The corresponding intermediate point P_{R} or P_{L} is determined such that, at this location, a tangential line segment connects $\mathbf{q}_1$ and the circle on the right or left hand side of v_0. Then, a PPC curve is designed to connect $\mathbf{q}_0$ and P_{R} or P_{L}, using the proposed design methodology. The resulting path can be either $\widehat{\mathbf{q}_0 P_{\mathrm{R}}}$ plus $\overrightarrow{P_{\mathrm{R}}\mathbf{q}_1}$, or $\widehat{\mathbf{q}_0 P_{\mathrm{L}}}$ plus $\overrightarrow{P_{\mathrm{L}}\mathbf{q}_1}$.

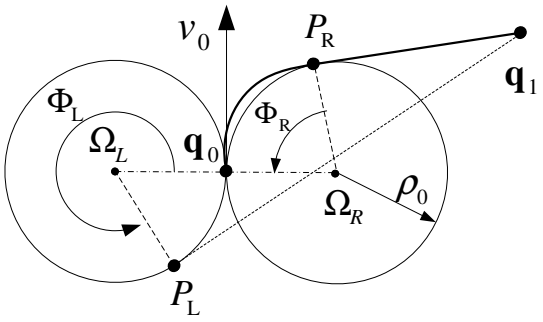


Figure 4.6: Combinations of a PPC curve and a line segment to connect $\mathbf{q}_0$ and $\mathbf{q}_1$ for a robot performing translation.

Half PPC Curve or Circular Arc for Turning Robot

If the robot is currently turning, the boundary conditions (4.1) are no longer satisfied, since $\kappa \neq 0$ at $\mathbf{q}_0$ now. One solution is to letting the robot continue turning around the current IC. That is, the circular arc centered at the current IC is to be used as the transition curve. In order for such a solution to exist, it is required that $\mathbf{q}_1$ must be on the circular arc. Apparently, this condition rarely holds.

Alternatively, we can utilize the second half part of a PPC curve (*half PPC curve* in short) and a straight line segment to connect $\mathbf{q}_0$, which is located at the middle of the PPC curve, and $\mathbf{q}_1$, as shown in Fig. 4.7. The half PPC curve is tangential to the orientation of $\mathbf{q}_0$. A line segment starting from $\mathbf{q}_1$ is tangential to a circle of radius ϱ_0 at point P_{R} or P_{L}. According to Eq. (2.8), the robot heading at $\mathbf{q}_0$ w.r.t. the polar axis can be expressed as

$$\gamma = \frac{\pi}{2} + \phi - \tan^{-1} \frac{\varrho'}{\varrho}|_{\phi=\frac{\Phi}{2}} = \frac{\pi + \Phi}{2}. \tag{4.20}$$

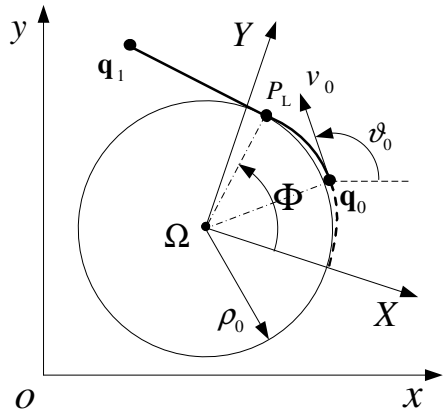


Figure 4.7: Combinations of half PPC curve and straight line segment to connect $\mathbf{q}_0$ and $\mathbf{q}_1$ for a turning robot.

If the angle of the polar axis w.r.t. the global frame is ϑ_X, it may be related to ϑ_0, the robot heading at $\mathbf{q}_0$, as follows:

$$\vartheta_0 = \pm(\pi + \Phi)/2 + \vartheta_X, \tag{4.21}$$

where the sign of $(\pi + \Phi)/2$ is determined by whether the polar frame is established to be CCW or CW.

Utilizing Eqs. (2.9) and (4.21), the polar coordinates of any point (ϱ, ϕ) on the half PPC curve can be transformed into the global cartesian coordinates as follows:

$$\begin{aligned} x &= x_\Omega + \varrho \cos(\pm\phi + \vartheta_0 \mp \tfrac{\pi+\Phi}{2}) \\ y &= y_\Omega + \varrho \sin(\pm\phi + \vartheta_0 \mp \tfrac{\pi+\Phi}{2}). \end{aligned} \tag{4.22}$$

For the point P_{L} or P_{R}, denoted here as P, we know that $\varrho(\Phi) = \varrho_0$. Thus, we have:

$$\begin{aligned} x_P &= x_\Omega + \varrho_0 \cos(\vartheta_0 \mp \tfrac{\pi-\Phi}{2}) \\ y_P &= y_\Omega + \varrho_0 \sin(\vartheta_0 \mp \tfrac{\pi-\Phi}{2}). \end{aligned} \tag{4.23}$$

According to Eq. (4.2), we know that $\varrho(\Phi/2) = \varrho_0(1 + \Phi^2/32)$. At the configuration $\mathbf{q}_0$, we have:

$$\begin{aligned} x_0 &= x_\Omega \pm \varrho_0(1 + \tfrac{\Phi^2}{32}) \sin\vartheta_0 \\ y_0 &= y_\Omega \mp \varrho_0(1 + \tfrac{\Phi^2}{32}) \cos\vartheta_0. \end{aligned} \tag{4.24}$$

As the line $\overline{\mathbf{q}_1 P}$ is perpendicular to the line $\overline{\Omega P}$ and $(x_P - x_\Omega)^2 + (y_P - y_\Omega)^2 = \varrho_0^2$, we have:

$$(x_1 - x_\Omega)^2 + (y_1 - y_\Omega)^2 = (x_1 - x_P)^2 + (y_1 - y_P)^2 + \varrho_0^2. \tag{4.25}$$

There is a constraint about ϱ_0 and Φ as in (4.10), where $\kappa|_{\phi=\frac{\Phi}{2}}$ is now the curvature of the path when the robot is at $\mathbf{q}_0$, and can be obtained using (2.33). It is noted that x_Ω, y_Ω, x_P, and y_P in Eqs. (4.23), (4.24) and (4.25) are unknown, and ϱ_0 and Φ are to be evaluated. With some necessary mathematical operations, we have the following constraint about Φ:

$$\begin{aligned} &(x_1 - x_0)\sin\left(\vartheta_0 \pm \tfrac{\Phi}{2}\right) - (y_1 - y_0)\cos\left(\vartheta_0 \pm \tfrac{\Phi}{2}\right) \\ &= \pm\tfrac{32(48+\Phi^2)}{\kappa_0(32+\Phi^2)^2}\left[1 - \tfrac{32+\Phi^2}{32}\cos\tfrac{\Phi}{2}\right]. \end{aligned} \tag{4.26}$$

By geometry, there will be at most one solution of P_L or P_R, respectively, and thus ϱ_0, and Φ. As such, bisection, secant, or inverse quadratic interpolation methods can be used to find the zero (if any) of the function (4.26).

The maximum value of $v(t)$ occurs at $\phi = \Phi/2$, i.e. at the location of $\mathbf{q}_0$. Therefore, according to Eq. (4.8), the duration of the motion along the entire PPC curve, can be given as follows:

$$\tau = \left(1 + \frac{\Phi^2}{32}\right)\frac{\varrho_0\Phi}{v_0}. \tag{4.27}$$

Remark 4.2. *Under the velocity profile (4.3), the velocity of a robot which moves along the curve does not change significantly. The design of combining a (half) PPC curve with a line segment allows the steering method to adjust the robot velocity when it moves along the line segment.*

4.2 Collision Test of PPC Curve for Path Generation

This section presents a simple yet time-efficient way for collision test of PPC curve in order for path generation/planning applications.

4.2.1 Collision Checking for PPC Curve

Lemma 4.1. *A PPC curve is concave, viewed from the polar origin.*

The curvature of a PPC curve is non-negative, as it is given by Eq. (2.11) and the following holds:

$$\varrho^2 - \varrho\varrho'' = \varrho(\varrho - \varrho\varrho'') = \varrho\varrho_0 \left(\frac{3}{2} + \frac{(\Phi-\phi)^2\phi^2}{2\Phi^2} - \frac{6(\phi-\Phi/2)^2}{\Phi^2} \right) \geq 0. \tag{4.28}$$

A curve is convex or concave, if the curvature along the curve has the same sign[3]. This implies that a PPC curve has a single concavity (which is concave, viewed from the polar origin) within the polar angle range $[0, \Phi]$.

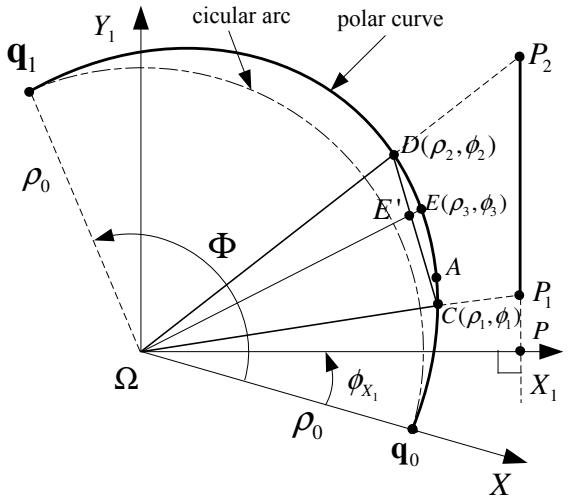


Figure 4.8: Collision checking between obstacle line segment and PPC curve.

According to the definition of concavity[4], we can come to the following corollary, provided that a PPC curve is concave w.r.t. the polar origin.

Corollary 4.1. *The straight line segment formed by any two points on a PPC curve is within the curve w.r.t. the polar origin, and does not intersect the curve other than the two points.*

As an obstacle can be regarded as a polygon consisting of line segments, collision test between a line segment and a PPC curve is studied in this section. The following pre-processing, without changing the result of collision test, is made: if a line segment

[3]The neighboring points at a curve of a point has the same concavity of that of the circle of curvature.

[4]If the curve of a function f always remains above the tangent lines for every point in the interval I, we say that the curve is concave upward on that interval.

intersects straight line $\overrightarrow{\Omega(\mathbf{q}_0)}$ or $\overrightarrow{\Omega(\mathbf{q}_1)}$, the end(s) of the line is(are) replaced by the corresponding intersection point(s). Suppose that the such obtained line segment is expressed as $\overline{P_1P_2}$, as shown in Fig. 4.8. The possible range of the two polar angles of the two end points must be either $\phi_1, \phi_2 \in [0, \Phi]$, or $\phi_1, \phi_2 \ni [0, \Phi]$. In the latter situation, there is apparently no intersection between the line and the curve. When $\phi_1, \phi_2 \in [0, \Phi]$, there can be a further categorization into the following three cases for analysis:

(i) If the two end points are not on the same side of the PPC curve, i.e.

$$|\Omega P_i| \leq \varrho(\phi_i), |\Omega P_j| \geq \varrho(\phi_j), i, j = 1, 2, i \neq j,$$

the obstacle line segment intersects the PPC curve, as the PPC curve completely separates the two regions on the two sides.

(ii) If the two end points are on the inner side of the PPC curve, i.e.

$$|\Omega P_i| < \varrho(\phi_i), i = 1, 2,$$

there will be no intersection according to corollary 4.1.

(iii) Otherwise, i.e. the two end points are on the outer side of the PPC curve, i.e.

$$|\Omega P_i| > \varrho(\phi_i), i = 1, 2 :$$

As shown in Fig. 4.8, a coordinate frame $O_1X_1Y_1$ is established with its origin located at Ω, and its axis X_1 being perpendicular to $\overrightarrow{P_1P_2}$. At point P, axis X_1 intersects the infinite straight line passing through points P_1 and P_2. By geometry, we can see that $\overrightarrow{P_1P_2}$ intersects the PPC curve if and only if the maximum value of the PPC curve segment ($\phi_1 \leq \phi \leq \phi_2$) projected to axis X_1 is not smaller than that of the line segment.

Let $X_1(\cdot)$ denote the X_1 coordinate of the point given in the bracket, ϕ_{X_1} denote the angle of axis X_1 w.r.t. the polar axis, A denote the point which, among all the points on the PPC curve segment, has the maximum X_1 value. The condition for the existence of an intersection can be expressed as

$$X_1(\mathrm{A}) = \max_{\phi \in [\phi_1, \phi_2]} [\varrho \cos(\phi - \phi_{X_1})] \geq X_1(P). \tag{4.29}$$

As the function $\varrho\cos(\phi-\phi_{X_1})$ is continuous, it has extremum values at the roots (if any) of its derivative and the boundaries $\phi=\phi_1$ and $\phi=\phi_2$. By comparing all the extremum values, the maximum value can be obtained. The derivative of the function is

$$\begin{aligned}\frac{d[\varrho\cos(\phi-\phi_{X_1})]}{d\phi} &= \varrho_0\left(\phi-\frac{3\phi^2}{\Phi}+\frac{2\phi^3}{\Phi^2}\right)\cos(\phi-\phi_{X_1}) \\ &\quad -\varrho_0\left(1+\frac{\phi^2}{2}-\frac{\phi^3}{\Phi}+\frac{\phi^4}{2\Phi^2}\right)\sin\phi,\end{aligned} \tag{4.30}$$

where the approximate numeric roots within $[\phi_1,\phi_2]$ can be obtained using secant method, or inverse quadratic interpolation method, or method of false position etc.[5]. In the case that f is (4.30), ϕ_1,ϕ_2 can be chosen as the initial values since the range $[\phi_1,\phi_2]$ is relatively small and contains the root (according to the meaning of X_1(A)).

4.2.2 Collision Checking for PPC Ray

Collision checking should be done between an obstacle and the surface swept by a robot due to the existence of the robot dimensions. When a rectangular robot follows a circular arc or a PPC curve, two curves are used to define its reach area, as shown in Fig. 4.9: "inner curve", combination of circular arc or PPC curve (radius $(\varrho_0-|CD|/2)$, center angle Φ_1) starting from B_0 and ending at B_1, and line segment $\overline{B_1G_1}$; and "outer curve", combination of line segment $\overline{A_0F_0}$, and circular arc or PPC curve (radius $(\varrho_0+|CD|/2)$, center angle Φ_2) starting from F_0 and ending at F_1. $\Phi_1=\Phi_2=\Phi$. The surface swept by the robot is then obtained by the sum of the following three items: i) the region (ray) enclosed by the inner and outer curves as well as line segments $\overline{B_0A_0}$ and $\overline{G_1F_1}$; ii) the rectangle $C_0D_0B_0A_0$; and iii) the triangle $G_1E_1F_1$. Collision checking for the last two items can be done by checking intersection between two line segments.

There remains the problem of collision checking between the ray and the obstacle. The same pre-processing as before is first made about each obstacle line segment without changing the result of collision test. The possible range of the two polar angles of $\overline{P_1P_2}$ can only be either $\phi_1,\phi_2\in[0,\Phi_3]$, or $\phi_1,\phi_2\ni[0,\Phi_3]$ (no intersection in this situation). A polar frame is established with its origin located at Ω, and the polar axis coinciding with line $\overrightarrow{\Omega(F_1)}$. Let functions $\varrho_{\text{in}}(\phi)$ and $\varrho_{\text{out}}(\phi)$ compute the

[5]The method of false position can be used in this research considering that it guarantees a solution if it exists.

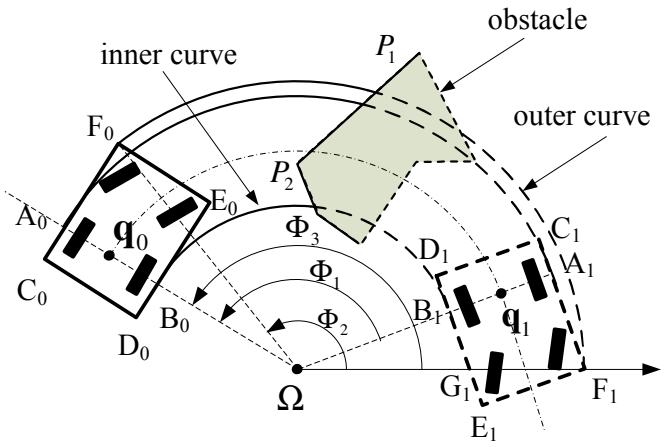


Figure 4.9: Surface swept by a rectangular robot.

distance between Ω and the point which has a polar angle ϕ, and is on the inner curve and the outer curve, respectively. When $\phi_1, \phi_2 \in [0, \Phi_3]$, there are three different cases to be analyzed:

(i) No intersection if the ranges of the two end points from the polar origin are within the inner curve, i.e.

$$|\Omega P_i| < \varrho_{\text{in}}(\phi_i), i = 1, 2.$$

(ii) If the ranges of the two end points from the polar origin are outside the outer curve, i.e.

$$|\Omega P_i| > \varrho_{\text{out}}(\phi_i), i = 1, 2,$$

the obstacle line segment intersects the ray if and only if it intersects the line segment $\overline{A_0 F_0}$ or the outer arc or PPC curve. Intersection checking with a PPC curve can be done using the method in Sec. 4.2.1. For the case of a circular arc, intersection checking can be easily done.

(iii) Otherwise, there exists an intersection, either the two end points are located on different sides of the ray, i.e.

$$|\Omega P_i| < \varrho_{\text{in}}(\phi_i), |\Omega P_j| > \varrho_{\text{out}}(\phi_j), i, j = 1, 2, i \neq j,$$

or at least one of the them is located within the ray, i.e.

$$\exists i \in [1, 2] : \ \varrho_{\text{in}}(\phi_i) \leq |\Omega P_i| \leq \varrho_{\text{out}}(\phi_i).$$

4.2.3 PPC Based Path Generation Algorithm

In the presence of obstacles, collision checking between obstacles and the PPC curve is carried out as previously stated. Before a collision test is carried out, range data obtained from laser rangefinders are filtered to exclude obviously erroneous ones, and then stored using the vector representation. In addition, the vector presentation is transformed into obstacle points, and each adjacent pair of obstacle points is assumed to form an obstacle line segment.

Algorithm 4 Con2Cnfgs($\mathbf{q}_0, \mathbf{q}_1, v, \omega$)

1: $turn\_dir \leftarrow$ direction of $\overrightarrow{\mathbf{q}_0\mathbf{q}_1}$ w.r.t. ϑ_0
2: $k_\varrho \leftarrow 1.2$, $n_\varrho \leftarrow 0$
3: **if** $\omega \approx 0$ **then**
4: TryRho: $\varrho_0 \leftarrow k_\varrho r_{\min}$, $n_\varrho \leftarrow n_\varrho + 1$, Obtain Ω, Φ, P
5: **if** $n_\varrho < n_{\varrho,\,\max}$ and constraint (4.17), (4.14), or (4.15) is not satisfied **then**
6: Adjust k_ϱ according to the type of violation; *go to* TryRho
7: **end if**
8: **else** ▷ Robot is rotating.
9: $turn\_dir \leftarrow$ direction of ω; $r \leftarrow v/|\omega|$, $\kappa_0 \leftarrow 1/r$; $\Omega_C \leftarrow \mathbf{q}_0, r, turn\_dir$
10: **if** $|\Omega_C\mathbf{q}_1| = r$ **then** ▷ Circular arc is used for connection.
11: $\tau \leftarrow \angle(\overrightarrow{\Omega\mathbf{q}_1}, \overrightarrow{\Omega\mathbf{q}_0})/|\omega|$
12: **else** ▷ Design a half polar curve.
13: Obtain Φ (solving (4.26)), ϱ_0 (4.10), Ω (4.24), P (4.23), τ (4.27)
14: **end if**
15: **end if**
16: **if** collision detected **then**
17: **if** Translation and $turn\_dir$ = direction of $\overrightarrow{\mathbf{q}_0\mathbf{q}_1}$ w.r.t. ϑ_0 **then**
18: $turn\_dir \leftarrow$ opposite of $turn\_dir$; *go to* Line 4 of this procedure
19: **end if**
20: *report* failure
21: **end if**
22: $t \leftarrow 0$ (polar curve) or $t \leftarrow \tau/2$ (half polar curve)
23: **for** $t < \tau; t = t + t_L$ **do**
24: Compute ϕ (4.3), ϱ (4.2), $v(t)$ (2.12), $\kappa(t)$ (2.11), $\omega(t) \leftarrow v\kappa \cdot turn\_dir$
25: enqueue (MotionCmds, $v(t), \omega(t)$)
26: **end for**
27: enqueue (MotionCmds) if includes a line segment

Algorithm 4 describes the procedure of connecting two configurations with a PPC curve and a line segment. The procedure searches the (ϱ, Φ) space to find a transition curve that satisfies the smoothness and feasibility requirements as well as the boundary conditions. If any of (4.14), (4.17), or (4.15) is not satisfied, the maximum velocity or the maximum curvature on the designed path will exceed the allowed one. If the maximum angular velocity or curvature exceeds the allowed one, we can try enlarge ϱ (and thus Φ) by increasing the value of k_ϱ, such that this maximum value

can be reduced. If the maximum velocity exceeds the allowed one, we may try decreasing the value of k_ϱ. Note that varying ϱ will change both the curvature and velocity profile of the PPC curve. If a curve is found, the algorithm checks whether there is any collision between the curve and the obstacles. It also tries the PPC curve with a different turning direction when no solution is found under the preferred one.

Remark 4.3. *The proposed method of collision checking is in general relatively effective in the following senses:*

(i) Actual collision test between a line segment and a PPC curve is done only when a collision is "really" possible – only if $\phi_1, \phi_2 \in [0, \Phi]$ and the two end points are on the outer side of the PPC curve. Otherwise ($\phi_1 \ni [0, \Phi]$ or $\phi_2 \ni [0, \Phi]$), or case ii) or case iii) of Sec. 4.2.1, only 2 times of comparisons of polar angles or polar radii are needed to get the result of collision test;

(ii) Collision test is transformed into the problem of root finding. We may choose a fast algorithm of root finding which requires the evaluation of the function (rather than its derivative); and

(iii) Actual collision test for a PPC curve is done only on a part (ϕ_1, ϕ_2) of the PPC curve rather than the entire curve (though the root found corresponds to the "deepest state" of collision out of the two possible intersections between a line and a PPC curve).

The proposed method of collision checking can be applied to the case of half PPC curve/ray, since the two types of curves share same properties in nature.

Remark 4.4. *Since the proposed method relies on Lemma 4.1 for collision test, it can be known that the method is in general applicable to any curve that has a single concavity or whose curvature is of the same sign.*

4.3 Hybrid Path Planning for Differential Drive Mobile Robots

This section presents a sensor-based hybrid approach for planning smooth, feasible paths for differential drive robots, by switching between instant goal generation and a fuzzy controller for wall following.

4.3.1 Approach Overview

Instant Goal (IG) is a point serving as a temporary goal for regulating a robot to keep moving along an obstacle. In the work presented in Chapter 3, IGs are determined for a holonomic robot without considering the robot dynamics or the requirement of curvature continuity. This however may result in the generation of IGs and paths that are non-feasible, and thus cannot be reached by a robot subject to nonholonomic and dynamic constraints.

Algorithm 5 Navigation to the Goal

```
1: Initialize S, G; i ← 0; mode ← MoveToG; Subscribe to laser messages
2: Accelerate R to a certain speed and move forward
3: MotionCmds=Con2Cnfgs(q_0, G, v, ω, PolarCurveOnly)                ▷ Algorithm 4
4: Turn until MotionCmds is Empty; mode ← MoveToG
5: while G is not reached do
6:     if laser messages are ready then
7:         Transform LaserData into obstacle points
8:         if mode = MoveToG then
9:             Move directly toward G
10:            if Meet an obstacle then
11:                mode ← FollowObs; decide SchDir; i ← i + 1; k ← 0
12:            end if
13:        else                                                      ▷ mode = FollowObs
14:            if queueIsNotEmpty(MotionCmds) then
15:                Move/Turn at velocity of dequeue(MotionCmds)
16:                if leave condition is satisfied then
17:                    mode ← MoveToG
18:                end if
19:            else
20:                k ← k + 1; S_{i,k} = P_t; Adjust velocity          ▷ Chapter 4.3.2
21:                if failure=SearchIG(v, ω, S_{i,k}) then
22:                    Generate P_f(d_tran, θ_rot) by fuzzy controller   ▷ Chapter 4.3.4
23:                    MotionCmds=Con2Cnfgs(q_0, P_f, v, ω)
24:                end if
25:            end if
26:        end if
27:    end if
28: end while
```

This section presents a path planning approach which locates IGs and paths satisfying the various robot constraints and the smooth motion requirement. Similar to the work presented in Chapter 3, it employs a Bug-like strategy to switch between the two motion modes in order to lead the robot to the goal. Algorithm 5 describes the procedure of navigating $\mathcal{R}$ to G from its initial position S. $\mathcal{R}$ first moves forward for a relatively short distance, and performs a turning so that it faces directly to G. Then, $\mathcal{R}$ decides either to follow an obstacle or to move directly toward the goal, based on

whether there is any obstacle blocking the robot from the goal. While following the i^{th} obstacle, $\mathcal{R}$ is guided to reach a series of IGs, $G_{i,k}, k = 1, 2, \cdots$, until a certain leave condition (e.g. the basic leave condition, see Chapter 3) is satisfied.

Each IG is determined by Algorithm 6 *SearchIG* (see Chapter 4.3.3). However, sometimes it might not be able to locate a proper IG, either because it has not searched the entire space or because the module takes too much time. The robot is thus unable to plan subsequent motions, which may lead to the failure of the operation of the whole autonomous system. Considering that a fuzzy controller can be robust in producing a motion as long as there is an arrival of sensory input, the above limitation of the path planner may be overcome by introducing a motion planned by a fuzzy controller for wall following. In the proposed approach, reactive motion planning is incorporated into path planning in a complementary way: a fuzzy reactor generates a number of motions when deliberate planning fails to, as shown in Algorithm 5.

For the purpose of better collision avoidance, the robot velocity is adjusted according to its surroundings before determining an IG or making a movement (Sec. 4.3.2). This is to ensure that the robot moves at a suitable speed while keeping a safe distance from obstacles.

Remark 4.5. *At the start of boundary following, the robot locates an IG and plans a number of motions toward it. The task of boundary following is achieved by locating a series of IGs. As such, the selection of IGs affects the path that the robot is to follow. As each IG is planned based on limited information obtained from online sensors, the proposed approach has yet to take into account global optimality of the path, which is one of the performance targets considered by many search-based path planning algorithms.*

4.3.2 Velocity Adjustment

The system makes a maintenance of a safety zone, and a safety-buffer zone (or buffer zone for short) around the forward path of the robot such that the robot will always be able to halt in time to prevent collisions. The safety zone $Z_{safe} \in \mathbb{R}^2$ is defined as the minimum obstacle free region that is required for the robot to stop (i.e. when maximum deceleration $a_{\mathrm{d,\ max}}$ is applied), given its current velocity, say v. The range of the safety zone is then given as follows, if sensing uncertainties are considered:

$$R_{safe} = \max(v^2, v_{\min}^2)/(2a_{\mathrm{d,\ max}}) + R_{\mathrm{rob}} + \sigma_S, \tag{4.31}$$

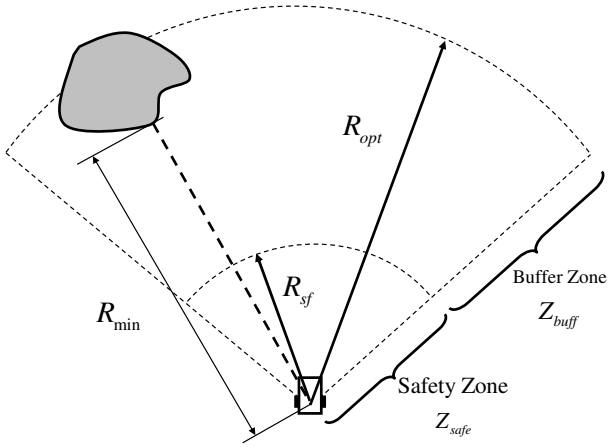


Figure 4.10: Safety and buffer zones of a robot.

where $\max(v^2, v_{\min}^2)$ is used instead of v^2 by taking into account the minimum velocity, $v_{\min}$, that the robot is able to produce.

The safety-buffer zone $Z_{buff} \in \mathbb{R}^2$ is a user defined buffer region that extends the safety zone Z_{safe} outward viewed from the robot, as shown in Fig. 4.10. The range of the buffer zone is the minimum distance (relative to the safety zone) required for the robot to reach its maximum allowed speed, given its current velocity:

$$R_{buff} = R_{safe} + (v_{\max}^2 - v^2)/(2a_{\text{s, max}}), \tag{4.32}$$

where $a_{\text{s, max}}$ is the maximum acceleration /deceleration that the robot is able to be accelerated/deccelerated comfortably and smoothly.

Before determining an IG or making a movement, the robot velocity is adjusted if needed. The robot is to slow down in the presence of obstacles around it. If the obstacles present are within the safety zone Z_{safe}, the robot is slowed down by applying the maximum permissible deceleration that does not cause skidding to occur. When the front of the robot is clear of obstacles, the robot can be accelerated gradually up to its optimal velocity. Otherwise, the velocity will be adjusted according to the surroundings, where the resulting acceleration is made sure to be within the acceptable bounds on a smooth acceleration:

$$v_d = \begin{cases} \max(0, v - a_{\text{d, max}}\Delta t), & \text{if obstacle in } Z_{safe} \text{ or stop requested} \\ \min(v_{\max}, \sqrt{\frac{\mu g}{\kappa}}, v + a_{\text{s, max}}\Delta t), & \text{if no obstacles in } Z_{buff} \\ \min(f_v(R_{\min}, v), \sqrt{\frac{\mu g}{\kappa}}), & \text{otherwise,} \end{cases} \tag{4.33}$$

where $R_{\min}$, the minimum distance of the robot to the obstacles in the front of it, and

the function f_v, are defined as follows:

$$R_{\min} = \min_{|\theta_j| \in \pi/4} (1 + \sin\theta_j) R_j, \tag{4.34}$$

$$f_v(R_{\min}, v) = \begin{cases} \min(\frac{R_{\min}}{R_{buff}} v_{\max}, v + a_{\text{s, max}} \Delta t), & \text{if } \frac{R_{\min}}{R_{buff}} v_{\max} \geq v \\ \max(\frac{R_{\min}}{R_{buff}} v_{\max}, v - a_{\text{s, max}} \Delta t), & \text{otherwise.} \end{cases} \tag{4.35}$$

In order for the robot to achieve a smooth and (actuator) feasible motion, the desired angular velocity is also limited to the following range:

$$\max(-\omega_{\max}, \omega - \varepsilon_{\max} \Delta t) \leq \omega_d \leq \min(\omega + \varepsilon_{\max} \Delta t, \omega_{\max}), \tag{4.36}$$

where $\varepsilon_{\max}$ is the maximum rotational acceleration of the robot.

In addition, we note that the current speed of the robot may prevent any suitable IG from being found. For such cases, the set of points reachable by the robot can be expanded through a reduction in velocity, which enhances the probability of finding an IG.

4.3.3 Deliberative Planning: IG Locating

Algorithm 6 describes the procedure of locating the IG $G_{i,k}$ (and generating a series of motions). The algorithm first computes $SchFrom_{i,k}$ and $\Theta_{i,k}$, using the procedure *GetSearchRange*($G_{i,k-1}$) (if $k = 1$, $G_{i,k-1}$ is undefined but any value passed to the function will make it work). Note that this step is exactly the same as that of the Instant Goal approach.

The algorithm then computes the IG* set $X_{ig,act}$. Within the search range Θ, not all the points can be taken as an IG*, since some areas are not reachable for the robot due to the dynamic constraints. In order to satisfy the constraints on acceleration (2.41) and velocity (2.42), the set of points, X_{rch}, that is reachable by the robot, should exclude the region within the two circles of the minimum turning radius, i.e. $r \geq r_{\min} = v^2/(\mu g)$. As shown in Fig. 4.11, the set of feasible IG*s is thus obtained as $X_{ig,act} = X_{rch} \cap \Theta$. This ensures that the IG*s are in the set of feasible solutions that satisfy the robot constraints, and that approaching an IG* within this set will keep the robot follow the obstacle in the desired direction.

Next, the algorithm computes IG* sampling region χ using the procedure *GetSamplingRegion*($X_{ig,act}, SchFrom_{i,k}, \Theta_{i,k}$), which checks each beam within $X_{ig,act}$ to decide a local sampling region in the beam's neighborhood. Let p_{near} and p_{far} denote

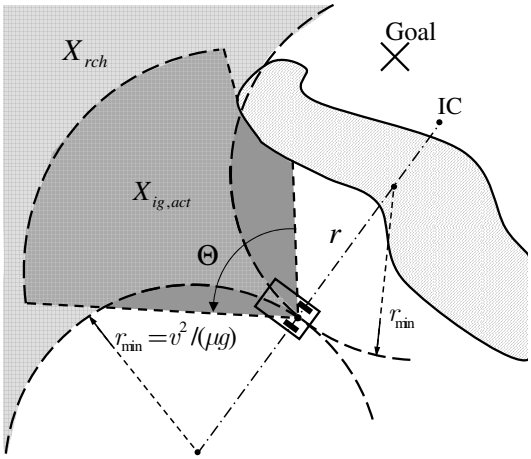


Figure 4.11: Determination of $X_{ig,act}$, the set of feasible IG*s.

the two obstacles points corresponding to the current beam and its next beam, with the subscript indicating whether a point is close to the robot or not. M is defined as the middle point of them. The local sampling region is then given as a disc, $\mathcal{C}(C, R_{\text{rob}}/2)$, located on the same side as the robot w.r.t. the line segment $\overrightarrow{p_{\text{near}}p_{\text{far}}}$. Point C is defined such that $\overrightarrow{MC} \perp \overrightarrow{p_{\text{near}}p_{\text{far}}}$ and $|MC| = R_{\text{sf}} + R_{\text{rob}}/2$. The final sampling region χ is then obtained by summing all the local sampling regions.

Algorithm 6 SearchIG$(v, \omega, \mathbf{q}_0)$

```
1: SchFrom_{i,k}, Θ_{i,k} ← GetSearchRange(G_{i,k-1})        ▷ Algorithm 2, Chapter 3.3.2
2: X_{ig,act} = X_{rch} ∩ Θ_{i,k}
3: χ =GetSamplingRegion(X_{ig,act}, SchFrom_{i,k}, Θ_{i,k})
4: for j_cur = SchFrom_{i,k} + 1; j_cur ∈ Θ_{i,k}; j_cur ← j_cur + 1 do
5:     Randomly sample a number of points ∈ χ around j_cur^th beam
6:     for each sampled point q_1 do
7:         if Succeed=Con2Cnfgs(q_0, q_1, v, ω) then
8:             return G_{i,k} and queue of MotionCmds
9:         end if
10:    end for
11: end for
12: report failure
```

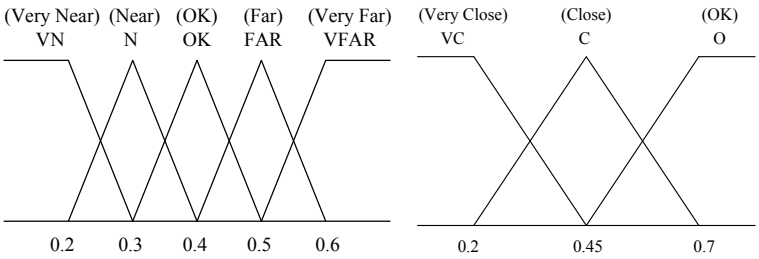
Finally, the algorithm searches for an IG with the priority of search given to the IG*s that are close (in the angular distance) to the direction of $SchFrom_{i,k}$. For the point currently sampled, the procedure $Con2Cnfgs$ (Algorithm 4) is used to test whether the two configurations can be connected by a combination of a PPC curve and a line segment.

4.3.4 Reactive Motion Planning: Fuzzy Wall Following

Expertise human knowledge is useful for wall following, and the natural linguistic language used in fuzzy logic would be able to capture this human sensing and intuitive reasoning. For example, a human controlling the robot makes decisions based on his perceptions of the obstacles, terrain and the goal and not by the mathematical analysis and models of the obstacles. Due to its robustness in producing a motion, a fuzzy controller for wall following is used to output a motion when no IG (and motion plans) can be found by the IG planner.

The design of such a fuzzy controller begins with the identification of the basic set of rules that provide guaranteed safe movements in dynamic unstructured environments. For wall following in the left direction of the robot (i.e. following an obstacle in the CW direction), the inputs are defined as the minimum range reading from the "right" ($-70°$ to $-90°$ w.r.t. the main axis) of the robot, d_{R}, and that from the "front" ($0°$ to $-20°$) of the robot, d_{F}. If the robot is to carry out wall following in the CCW direction, we only need to change the definitions of the two inputs accordingly and reverse the sign of the output angular velocity that is obtained as described later.

The two numerical inputs are fuzzified by the membership functions as shown in Figs. 4.12(a) and 4.12(b), respectively. These membership functions are defined based on the expert knowledge, and not directly related to the size of the robot; that is, the inputs are the distances of obstacles from the center of the robot body. To be free from collision, certainly the distances should be larger than the robot's effective radius. The two fuzzified inputs (represented by the fuzzy sets) are then passed into a fuzzy controller to generate the translational velocity v and the rotational velocity ω.



(a) Input: d_{R} (b) Input: d_{F}

Figure 4.12: Membership functions for input distances.

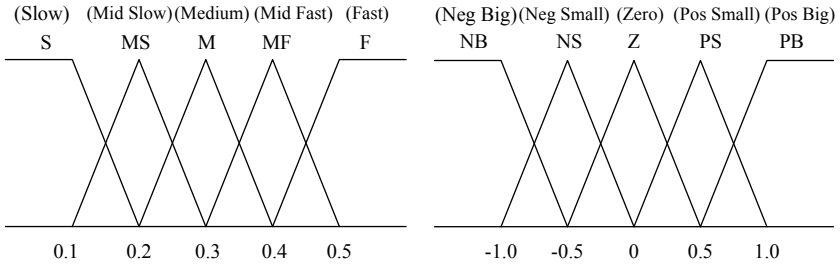
The fuzzy controller, consisting of 15 rules, is designed as in Table 4.2. The rules

are to allow the robot to keep a constant distance away from the obstacles in the right side of it. The values of d_{R} and d_{F} are regulated to keep the robot at a distance not too far away from the wall and in the meantime safe from collision with the wall. Due to the heuristic nature of a fuzzy system, each rule is determined by a trial and error basis according to the human expertise knowledge of driving a vehicle, until the desired results can be achieved. The two outputs of the fuzzy controller are fuzzified values of the translational and rotational velocities.

Table 4.2: Rules for Controlling Translational/Rotational Velocities.

d_{F} \ d_{R}	VN		N		OK		FAR		VFAR	
VC	S	PB	S	PB	S	PB	S	PB	S	PB
C	S	PB	MS	PS	M	PS	MS	NS	S	NS
O	MF	PB	F	PS	MF	Z	MF	NB	M	NB
	v	ω	v	ω	v	ω	v	ω	v	ω

In order to be able to serve as the velocity commands to drive the robot, the two outputs of the controller are defuzzified using the membership functions in Figs. 4.13(a) and 4.13(b), respectively. A negative ω indicates clockwise direction and vice versa. When designing the output membership functions, the translation and rotation velocities should be within their corresponding range that is confined by the maximum translation or rotation speed, respectively.



(a) Translational velocity v (b) Rotational velocity ω

Figure 4.13: Membership functions for output velocities.

The proposed approach is a hybrid between a path planner and a reactor, and generates a series of motions ahead of the execution of them. The output velocities of the fuzzy controller are then converted to a motion vector (d_{tran}, θ_{rot}), which includes translation displacement and angular displacement relative to the current robot pose. As illustrated in Algorithm 5, the function *Con2Cnfgs(*$\mathbf{q}_0, P_f, v, \omega$*)* is used to produce

a PPC curve plus a line to connect to P_f, determined by the motion vector (d_{tran}, θ_{rot}). If a series of motions cannot be planned, the approach then directly uses the output velocities of the fuzzy controller for reactive planning.

The switching to reactively planning motions is worthwhile, in that a non-smooth motion is robustly produced, and thus the robot may continue to move forward, rather than becoming stuck and failing completely to carry out the path planning task.

4.4 Simulation Experiments

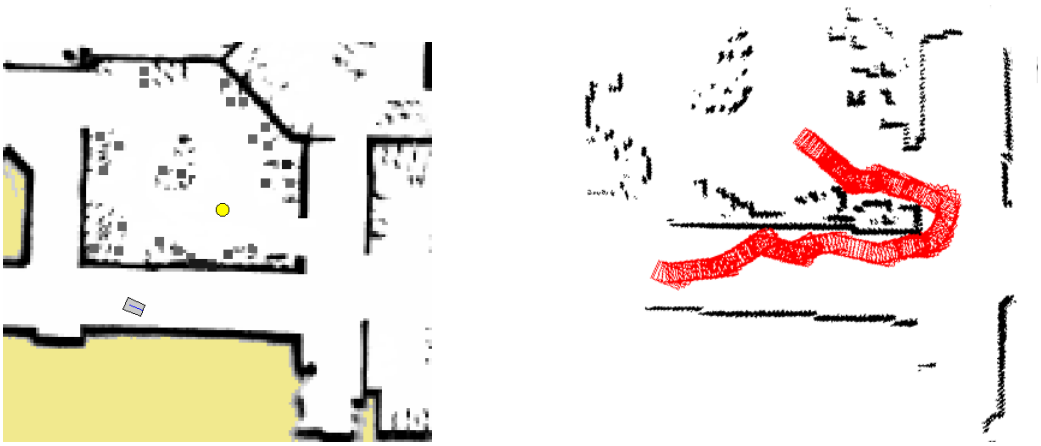
The proposed hybrid approach and the path generation algorithm were implemented within the CARMEN architecture (see Appendix C) in C programming language on a Linux operating system. In all the simulations, the same set of parameter values as shown below were used:

- Two laser rangefinders with detectable range 20 m, angular resolution is 1° (i.e. $N_s = 360$), and sampling rate 5 HZ, are mounted in the front and rear of the robot.
- Errors are introduced to sensory perceptions. Both range error variance and azimuth error variance are set as 0.1%. The probabilities of erroneously obtaining maximum and random range measurements are both set as 0.01%.
- A differential drive robot in rectangular shape was used to test the proposed hybrid path planning approach. Its dimensions are width $L_1 = 0.40$ m, length $L_2 = 0.60$ m, and $d = 0$ m. The robot dynamics is set as $v_{\max} = 0.5$ m/s and $\omega_{\max} = 2.0$ rad/s, $a_{\text{d, max}} = 1.2$ m/s^2, $a_{\text{s, max}} = 0.5$ m/s^2, and $\varepsilon_{\max} = 2.0$ rad/s^2.
- A car-like robot was used to test the path generation algorithm. Its dimensions are width $L_1 = 0.40$ m, length $L_2 = 0.60$ m, and $L = 0.45$ m. Minimum turning radius of the robot is $r_{\min} = 0.60$ m. The robot dynamics is set as $v_{\max} = 0.60$ m/s, $\omega_{\max} = 0.80$ rad/s, and $\dot{\varphi}_{\max} = 5$ rad/s, respectively.

4.4.1 Test on a Differential Drive Robot

A typical test of the proposed hybrid path planning approach was conducted in an environment as shown in Fig. 4.14(a), where the robot is plotted as a rectangle, its orientation as a line segment, and the goal as a circle. Fig. 4.14(b) shows the trajectories of the robot which navigated from left to right in the figure. After a slight movement, the robot performed rotations and turned left before moving directly

toward the goal. The robot made this turning because it does not have full omnidirectionality with simultaneous and independently controlled rotational and translational motion capabilities to change its orientation instantly. In contrast, many path planning approaches assume that the robot can be steered in any desired direction without the need of turning the robot first.



(a) Environment, initial pose of robot, and goal.

(b) Sequence of robot motions and laser scans obtained.

Figure 4.14: Initial pose and trajectories of robot.

The velocity profiles of the output path are shown in Fig. 4.15. The types of robot paths included "straight line", "PPC curve", "half PPC curve", "circular arc" and "other curve" (which is produced by the fuzzy controller). The velocities had a continuous change especially over the part of path that is a PPC curve or a half PPC curve. In contrast, the Instant Goal approach considers a holonomic robot that is able to move directly to a position with an abrupt change in its motion direction. In addition, it does not respect the bound limit of the curvature of path, and the robot can move along a path with the curvature being discontinuous.

For some PPC curve parts of the path, the angular velocity reached its maximum close to the maximum allowed value $\omega_{\max}$ while the linear velocity at that moment was much smaller than the allowed one. This can be explained by taking a look at Eqs. (4.14) and (4.17), when ϱ_0 is of a small value (e.g. 0.06), the angular velocity constraint (4.17) is easier to be violated under the current settings of $v_{\max}$ and $\omega_{\max}$. Therefore, in order for Algorithm 4 to find a solution curve, it will enlarge ϱ (and thus Φ) and reduce the maximum value of the angular velocity.

The linear velocity along the whole path was relatively low (average speed was a bit below 0.2 m/s), which may be explained by the effect of the dynamics on

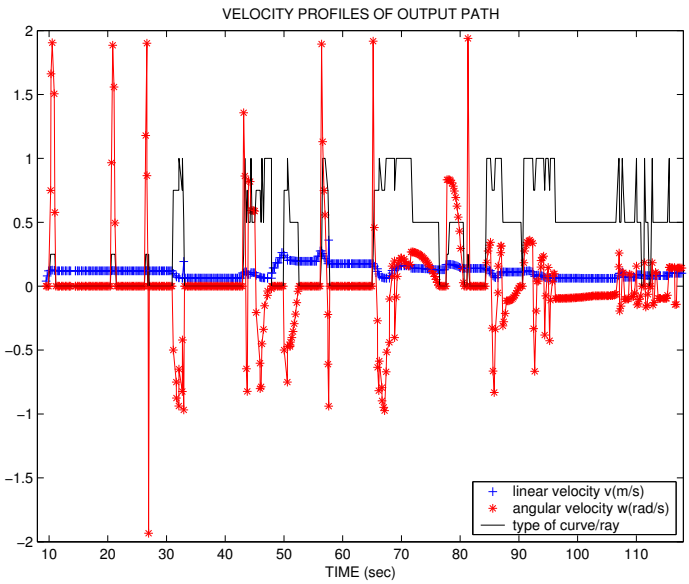


Figure 4.15: Velocity profiles of output path. Dash line denotes curve type (1: PPC curve, 0.5: half PPC curve, 0: line segment, and 0.75: other curve.)

robot safety in an obstacle-cluttered surrounding. As boundary following is normally carried out when the robot is relatively close to obstacles, the robot needs to move with limited speed in order to stay within the buffer zone as defined in Section 4.3.2 or farther from the obstacles. Fig. 4.15 shows that parts of the motion commands were generated through the fuzzy controller. It is illustrated that, when no IG (and motion plans) can be found by the IG path planner, it is able to produce a motion plan as long as there is an arrival of sensory input. The proposed approach thus incorporates reactive motion planning into path planning in a complementary way.

A robot in reality is not always able to exactly track the motion commands, due to constraints imposed by the robot dynamics. This may cause problems not only to the path/motion planning itself (i.e. the resulting path/trajectory could be very different from the expected ones), but also to the safety of the robot or the surroundings. Fig. 4.16 plots the "actual" velocities, which were generated through the simulator module that has modeled the dynamics of a differential drive robot. Compared to the result in Fig. 4.15, the actual velocities tracked the motion commands well when the path is relatively smooth, but the robot was unable to rapidly follow the angular velocity with a relatively big magnitude. This problem may be partially solved by designing a trajectory tracking control algorithm [102]. It is noted that the actual velocities became zero at some specific intervals. This may be because collision avoidance, having a higher priority, was occasionally triggered and thus prevented

motion commands from being executed.

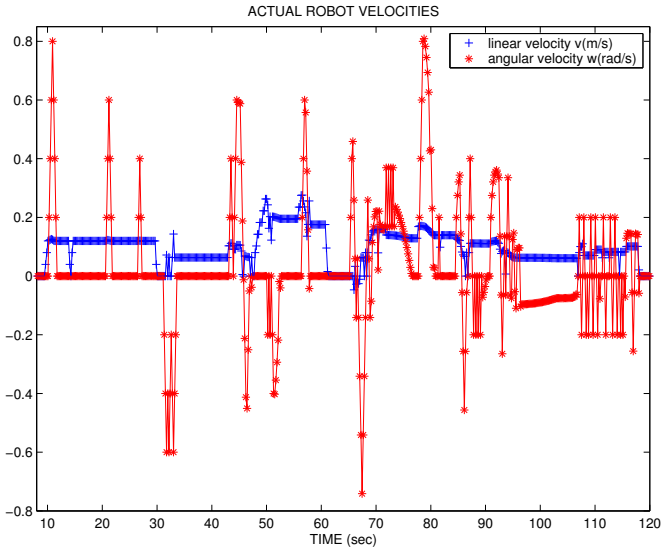


Figure 4.16: Actual velocities executed by the robot.

Fig. 4.17 shows the laser measurements and the robot trajectories obtained in another test. During the entire planning process, laser scans were continuously added to the plot without erasing the previous plotted scans. At the beginning, the robot chose the left direction to follow the obstacles, which led to its subsequent movement in the direction not optimal in the global sense and eventually entering the state of following the long corridor. Nevertheless, the robot was able to follow the obstacles in the desired direction. At the sharp turns when the robot entered or exited the U-shaped obstacles, the robot constantly needs to make a decision out of possible moving directions, and performed a series of rotations before successfully bypassing the sharp turns. These negotiations are attributed to the effects of the dynamics restrictions (that is, the robot's inability to make sharp turns), and the smaller probability in finding a solution combined by a PPC curve and a line segment. On the other hand, it is noted that the robot moved in the straight-line direction at a relatively fast speed when following a wall. It is also shown that at the places consisting of small corners, the robot was slowed down and made some turnings instead of moving in a straight-line direction. The robot tried to follow the obstacle's boundary without missing out possible passage at those moments.

The effect of a discrete sensor (i.e. the number of beam readings) on path planning results has yet to be studied through simulations using different settings of laser range angular resolution. However, the authors believe that the reduction of this resolution

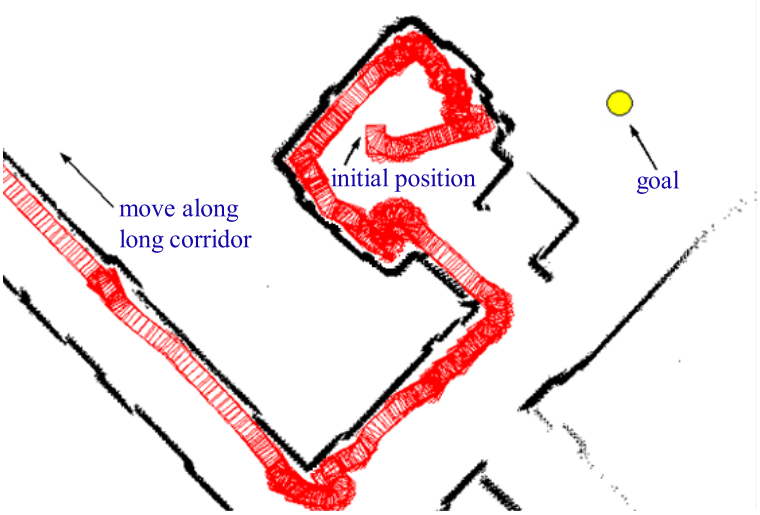


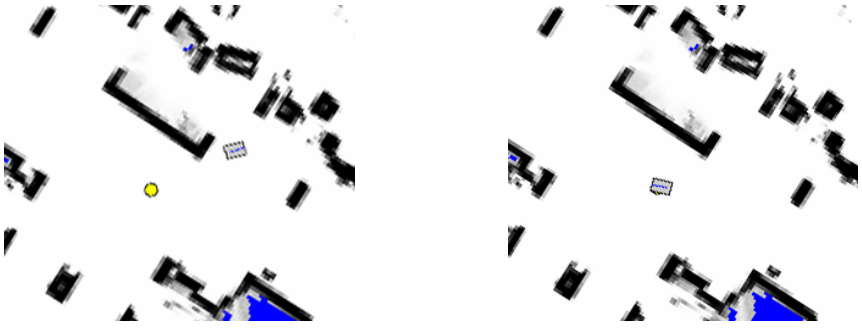
Figure 4.17: Sequence of robot motions and laser scans obtained in the second test.

(which is currently set as the commonly used one, i.e. 1°) is expected to decrease the probability of finding a suitable IG, as fewer beams will be searched.

4.4.2 Test on a Car-like Robot

Analysis of Paths Generated

A test of the path generation algorithm was conducted on the car-like robot in an environment as shown in Fig. 4.18(a). The robot is denoted by a rectangle and its orientation is denoted by a straight line. The goal is denoted by a circle. It can be seen that the robot was able to reach the goal at the final pose (Fig. 4.18(b)) by moving from its initial pose (Fig. 4.18(a)).



(a) Initial pose of the robot. (b) Final pose of the robot.

Figure 4.18: Initial and final (upon reaching the goal) poses of the robot.

Fig. 4.19 shows the laser measurements and the trajectories of the robot, where each robot pose is plotted as a rectangle. During the entire process, laser scans were continuously added to the plot without erasing the previous plotted scans. After moving a bit in straight line direction, the robot turned to right and moved toward the goal. This is due to that a car-like robot does not have full omnidirectionality, and the collision test, based on the range readings, detected a possible collision if turning to left.

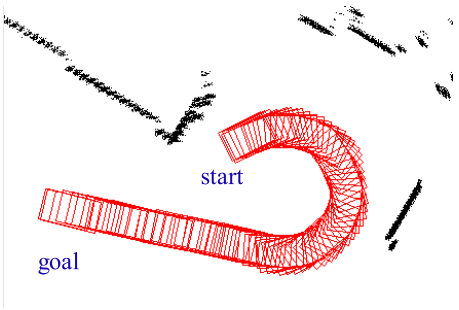


Figure 4.19: Sequence of robot motions and laser scans about the environment.

The curvature and velocity profiles of the output path are shown in Fig. 4.20. The curvature κ changed continuously throughout the entire path length, and its maximum value was around 1.28 m^{-1}, which is lower than the maximum limit (1/0.6) m^{-1}. The steering angle φ and the front-wheel velocity $\mathcal{V}$ changed continuously, and their magnitude increased or decreased in the trend similar to that of κ. These observations suggested that the proposed method is able to generate smooth paths and continuously-changing motion commands for a car-like robot.

Path Including Half PPC Curve

In another test, half PPC curve was used in addition to PPC curve. Fig. 4.21(a) plots the trajectories of the robot. The turning was found to be not a curve symmetrical to the polar angle $\phi = \Phi/2$, one of the properties that a PPC curve possesses. During navigation, an occupancy grid map of the environment was built online, as shown in Fig. 4.21(b).

The curvature and velocity profiles of the output path are shown in Fig. 4.22, where dash-dotted lines denote the curve types of the different segments. The transition from "PPC curve" to "half PPC curve" is because the laser range readings (can be very noised) obtained online imply potential collisions if continuing following

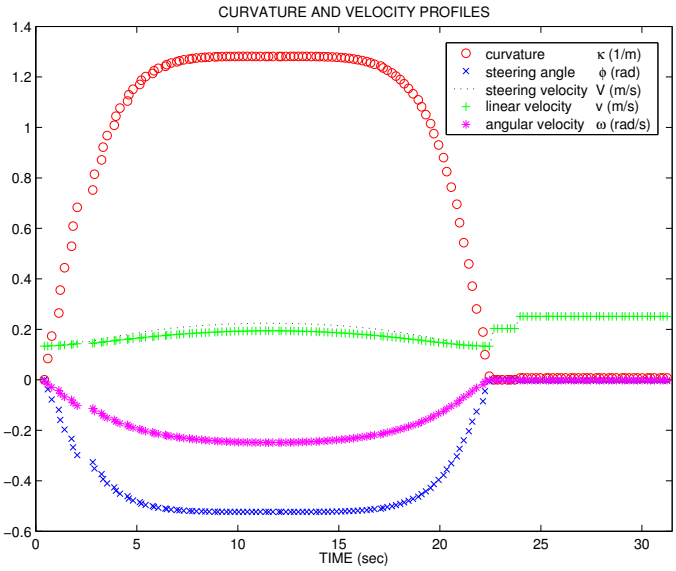
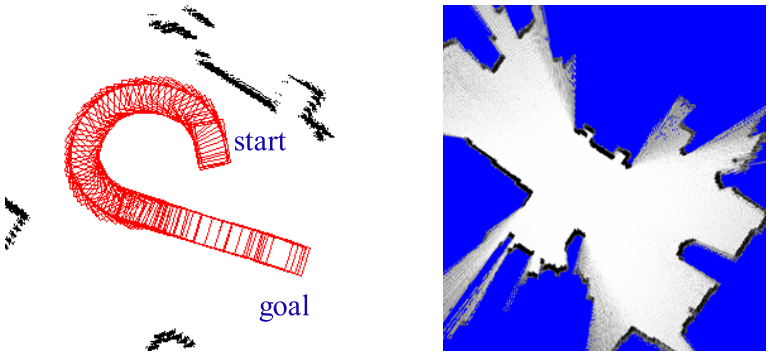


Figure 4.20: Curvature and velocity profiles of the output path.



(a) Robot motions. (b) Constructed grid map.

Figure 4.21: Robot trajectories and obtained grid map of the environment.

the path, and the robot was performing rotations at that time. It is shown that the curvature and velocities changed continuously at all the transition points, which suggested that the proposed method is capable of making a smooth connection when the robot performs either translation or rotation. For the PPC and half PPC curves, the maximum curvatures occurred at places other than $\Phi/2$ as $\Phi = 1.42\pi$ or 1.99π.

Analysis of Time for Path Generation

Fig. 4.23(a) shows t_L, relative laser time stamp between two continuous batches of laser readings, and the corresponding reaction time t_R used to generate a motion, in the first test on the car-like robot in Chapter 4.4.2. It is shown that the actual

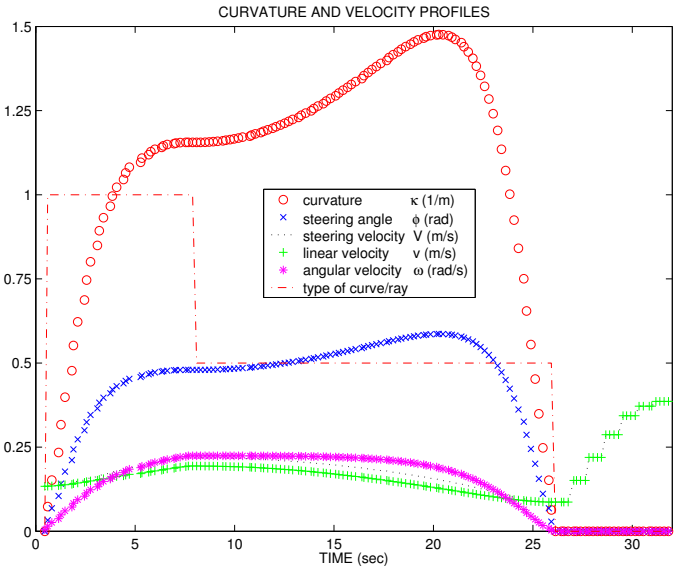
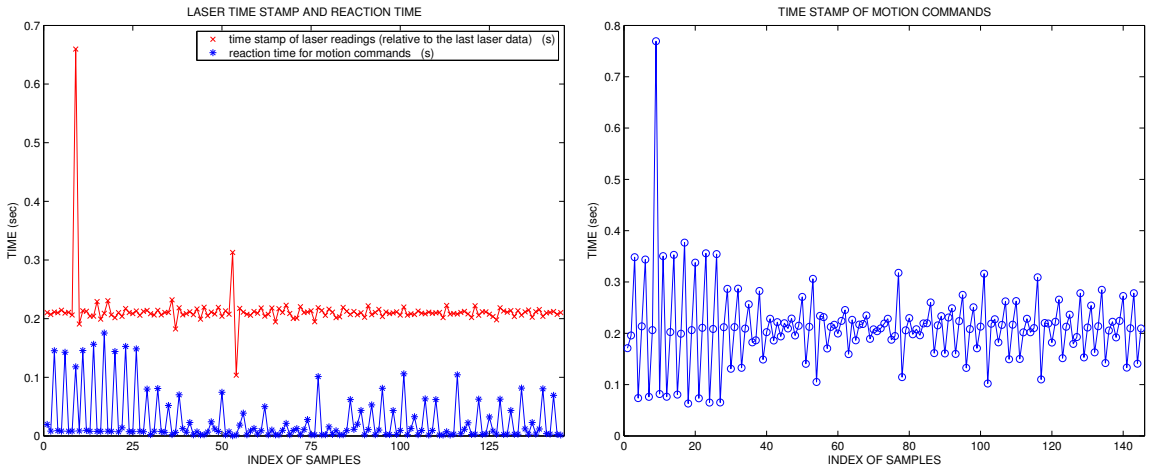


Figure 4.22: Curvature and velocity profiles and curve type of the output path.

time for generating laser data was generally slightly longer than 0.2 s, and at several sample periods, its value differed significantly from the average one. Fig. 4.23(b) shows that each reaction time spent to produce a motion command was within 0.2 s, and its average value was 0.0258 s. These observations suggest that the proposed approach, where PPC curve design and collision test are involved, is able to produce collision-free motion plans in real-time.



(a) Laser time t_L and reaction time t_R.

(b) Motion time stamp t_M.

Figure 4.23: Laser time stamp, and reaction time for motion commands.

Table 4.3 shows a statistics of the time used. The laser time stamp to generate the 9th laser data, 0.6597 s, was the maximum one, and was comparatively longer than

the required time, 0.2 s. Accordingly, the motion time stamp reached the maximum value 0.7693 s, at the 9th sample period. These phenomena can also be seen from Figs. 4.23 (a)-(b). They explain the big time difference between the two commands at the time of around 2.5 s in Fig. 4.20.

Table 4.3: Statistics of Time Used in Motion (145 samples in total, counted from the first motion command).

	Average Value (s)	Max Value (s)	Sample Index Max Occurs
Laser time stamp	0.2130	0.6597	9
Reaction Time	0.0258	0.1755	17
Motion time stamp	0.2126	0.7693	9

Laser time stamp and reaction time of the first test on a differential drive robot presented in Chapter 4.4.1 is shown in Fig. 4.24. Again, the actual time for generating laser data t_L was generally slightly longer than 0.2 s, and at several sample periods, its value differed significantly from the average one. It is shown that each reaction time spent to produce a motion command was within 0.2 seconds, and its average value was less than 0.05 s. Therefore, the algorithms for curve design and collision checking are suitable for real-time implementation.

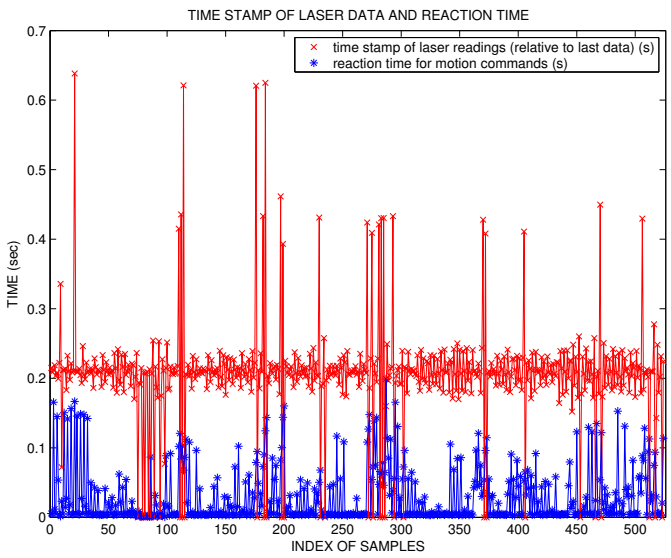


Figure 4.24: Reaction time for motion commands upon arrival of sensor data.

4.5 Discussions and Comparisons

The work in [22] presents sensor-based path planning algorithms which adopt a Bug-like strategy for navigating around an obstacle. They assume that the robot has at most six possibilities to move at a certain position, such that Best-first and Depth-first algorithms can be used for searching the state-space graph of the system's C-space and thus an optimal solution may be found by comparing search costs. In comparison, the proposed approach attempts to use a complex curve to connect the current position to a candidate local goal. Furthermore, it incorporates both the dynamic constraints and the curvature-continuous requirement into motion planning, rather than considering the nonholonomic constraints only.

The proposed IG planner uses a combination of PPC curve and line segment to connect the robot's current position with the IG, in order to plan a path with smooth velocity profile satisfying the robot dynamics constraints. In this way, it plans a series of smooth, feasible motions before actually executing them, and thus avoids unnecessary jerky manoeuvres that may happen in many purely reactive approaches such as potential field methods or fuzzy approaches of wall following [103]. However, sometimes it might not be able to locate a proper IG, probably because it has not searched the entire space (i.e. combinations of PPC curves and line segments are only part of the entire possible solution set to the problem of smooth path planning). Such cases hinder the robot from being able to plan subsequent motions, which may lead to failure of operation of the whole system.

In comparison, fuzzy approaches for wall following plan each motion in a purely reactive manner according to the local sensory input rather than planning paths deliberatively. Unlike the IG planner, such approaches seldom consider the requirement of smooth motion when generating motion commands. In addition, a simple implementation of such approaches is insufficient to ensure that the robot always follows an obstacle properly. However, considering its robustness in producing a motion whenever there is sensory input, a fuzzy controller is introduced to overcome the limitation of the IG planner. By combining the two kinds of planners in a complementary way, the proposed approach is thus a complete path planner.

The proposed PPC-based method is able to directly address dynamic and constraints during design, provides closed expression of robot position, and carries out collision test efficiently. However, a complex and parameterized curve only provides

a limited set out of the entire solution set for smooth path planning. The combined curve consisting of a PPC curve or a half PPC curve covers only part of the many possible feasible paths. When applied to path planning for car-like robots in a sensor-based scenario, the PPC based method magnifies its limitation that it may have a small probability in finding a suitable solution based on limited information. In Chapter 6, an optimization approach will be presented which searches the entire feasible velocity space for the best motion satisfying dynamic and other constraints.

4.6 Summary

This chapter has proposed a hybrid approach for planning smooth paths satisfying dynamic constraints for differential drive robots in an unknown environment, and a constrained, smooth path generation algorithm for car-like robots.

We first investigated the use of polar polynomial curve (PPC) for smooth, feasible vehicle motions between two arbitrary robot configurations. A computationally efficient method is proposed for collision checking of the designed curve, for it to be used in real-time path planning. The PPC-based steering method is applied to path generation for car-like robots, where the curvature of the path is required to be continuous and upper-bounded.

Then, we presented a hybrid planning approach to guide the robot to move forward along the boundary of an obstacle of arbitrary shape, leading the robot to the goal, by generating Instant Goals (and a series of deliberate motions) and planning reactively using a fuzzy controller when needed. In order to achieve smooth and feasible vehicle motions, the current robot position is connected with its local goal using PPC based curves.

Finally, simulation experiments have verified the effectiveness of the proposed approaches in planning collision-free, feasible paths for differential drive and car-like robots in a real-time manner. In addition, extensive discussions were provided on the effect of the robot dynamics on robot motions and on convergence to the goal.

Chapter 5

Online Map Building for Autonomous Mobile Robots

This chapter presents a practical method for building maps online with laser and sonar data fused to produce a better representation of the environment. Range data are translated into grid status information of the (local) map, which is then updated to the global map using Bayes' rules. To achieve the task of online mapping, incremental Maximum Likelihood scan matching is applied for online pose estimation before a local map can be updated to the global one correctly. Next, a selective method is proposed to fuse laser and sonar data for better obstacle detection and mapping. Then, the system is able to build maps autonomously with a fuzzy controller for wall following. Finally, an off-line method is implemented to convert the constructed occupancy grid map into a topological one that could be suitable for large map applications.

5.1 Incremental Map Building

5.1.1 Sensor Model

Compared with geometric or topological maps, occupancy grids provide more detailed information about the environment, and can be easily updated when there is a sensor input due to the probabilistic nature of grids. This research uses occupancy grid map for mapping and path planning in a non-large-scaled indoor environment. Occupancy grid is probabilistic in nature and inaccuracies in laser readings can be taken into account. As more laser scans are plotted down, the plot of laser scans becomes

cluttered as small errors in the laser readings cause the walls to become thicker and thicker. The use of occupancy grids allows to demarcate the occupied, unoccupied and unknown areas. In addition, it has a certain ability to handle dynamic environments, since objects which move during mapping tend to be erased off (because of the map updating process) in the map as more readings are taken in.

A sensor model must interpret (range) sensor data based on the sensor's characteristics and behavior. In the Gaussian sensor model, the sensor probability function can be given by

$$p(r|z,\theta) = \frac{1}{2\pi\sigma_r\sigma_\theta} \exp\left[-\frac{1}{2}\left(\frac{(r-z)^2}{\sigma_r^2} + \frac{\theta^2}{\sigma_\theta^2}\right)\right], \tag{5.1}$$

where r is the range reading, θ the angle with the optical axis of the range sensor, z the true space range value, σ_θ a measure of angular error, and σ_r a measure of range error. We may model the probability density of an range sensor with a Gaussian function multiplied by $\alpha(r)$, an attenuation of detection with distance, and added by a constant p_c, in a form similar to Eq. (3.35).

An angle measurement of a laser rangefinder can be represented by a normally distributed random variable whose 95% error bound is given by its angular resolution. In this case, if the angular resolution is given as $\Delta\alpha_s$, we have

$$2\sigma_\theta = \frac{\Delta\alpha_s}{2} \quad \Rightarrow \quad \sigma_\theta = \frac{\Delta\alpha_s}{4}. \tag{5.2}$$

A typical value of σ_θ would be 6 degrees (for a sonar sensor with a nominal beam width of 24 degrees), or 0.25 degree for a laser rangefinder. The range error σ_r of a Polaroid sonar sensor is on the order of 1%, and a conservative function we may use is: $\sigma_r = 0.01 + 0.015r$. For a laser rangefinder, range error is typically as small as $\pm$5cm.

To reduce computation load, a numerical function can be used to interpret sensor data into grid status. For each cell m_l in the sensor cone we compute the distance d from the current robot pose x_t. Let $N(\theta)$ denote a weighted Gaussian function $\frac{\alpha}{\sqrt{2\pi}\sigma_\theta} \exp\left(-\frac{1}{2}\frac{\theta^2}{\sigma_\theta^2}\right)$. In order to reduce the time and memory consumed in updating maps, the occupancy probability at the cell, $P(m_l|z_t, x_t)$, can be computed using an

approximated Gaussian model as follows:

$$P(m_l|z_t, x_t) = \\ P(m_l) + \begin{cases} -N(\theta), & d < z_t - \Delta d_1 - \Delta d_2 \\ \left[1 + 2\frac{d-(z_t-\Delta d_2)}{\Delta d_1}\right] N(\theta), & z_t - \Delta d_1 - \Delta d_2 \leq d < z_t - \Delta d_2 \\ N(\theta), & z_t - \Delta d_2 \leq d < z_t + \Delta d_3 \\ \left[1 - \frac{d-(z_t+\Delta d_3)}{\Delta d_4}\right] N(\theta), & z_t + \Delta d_3 \leq d < z_t + \Delta d_3 + \Delta d_4 \\ 0, & z_t + \Delta d_3 + \Delta d_4 \leq d, \end{cases} \tag{5.3}$$

where we can set $\alpha = 0.2$ in practice, and, considering that obstacles (such as a wall) have a certain width, $\Delta d_3 > \Delta d_2$. Fig. 5.1 shows such a model for sonar, where $\sigma_\theta = 0.1$ (6 degrees) and $\sigma_r = 0.06$m (the measured range is about 3m).

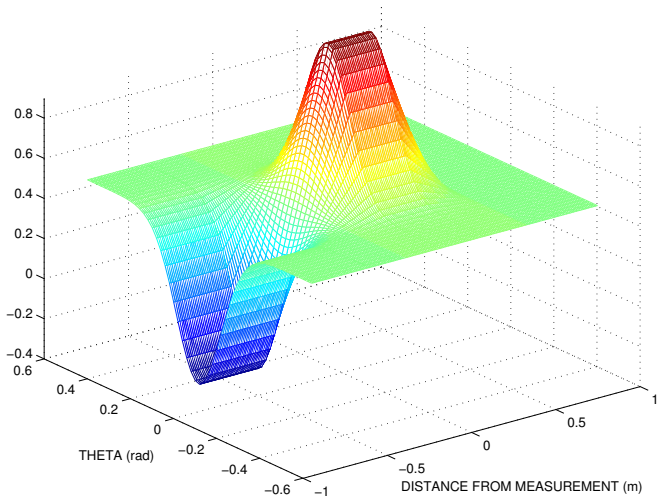


Figure 5.1: An approximated Gaussian sensor model, where "distance from measurement" means $d - z_t$ in Eq. (5.3) and $\Delta d_3 = 2\Delta d_2$.

5.1.2 Bayesian Map Updating

Bayes' Theorem [33] is chosen here considering its efficiency for updating a measurement to a grid map over other updating techniques, such as Dempster-Shafer theory of evidence [34, 35], and fuzzy logic approaches [36, 37]. Firstly, we assume that the robot knows its position and orientation while mapping the environment. Let $x_{1:t} = x_1, x_2, ...x_t$ denote the pose series of the robot at each time instant and $z_{1:t} = z_1, z_2, ...z_t$ the perceptions of the environment. The solution is to find the map $P(m|x_{1:t}, z_{1:t})$ expressed in the form of the current measurement. A number of works

have developed their own formula for updating the latest measurement to the global map. The following derivation is most similar to the one in [104].

If we apply Bayes rule given map m is known, we have

$$P(m|x_{1:t}, z_{1:t}) = \frac{P(z_t|m, x_{1:t}, z_{1:t-1})P(m|x_{1:t}, z_{1:t-1})}{P(z_t|x_{1:t}, z_{1:t-1})}. \tag{5.4}$$

Assuming measurement z_t is independent from the previous poses $x_{1:t-1}$ and measurements $z_{1:t-1}$, Eq. (5.4) can be rewritten as

$$P(m|x_{1:t}, z_{1:t}) = \frac{P(z_t|m, x_t)P(m|x_{1:t}, z_{1:t-1})}{P(z_t|x_{1:t}, z_{1:t-1})}. \tag{5.5}$$

For the item $P(z_t|m, x_t)$ in Eq. (5.5), again apply Bayes rule:

$$P(z_t|m, x_t) = \frac{P(m|z_t, x_t)P(z_t|x_t)}{P(m|x_t)}, \tag{5.6}$$

where $P(m|x_t)$ can be replaced by $P(m)$ considering that x_t does not carry information about the map m.

Substituting Eq. (5.6) into Eq. (5.5), we have

$$P(m|x_{1:t}, z_{1:t}) = \frac{P(m|z_t, x_t)P(z_t|x_t)P(m|x_{1:t}, z_{1:t-1})}{P(m)P(z_t|x_{1:t}, z_{1:t-1})}. \tag{5.7}$$

Considering that each cell $m(l)$ is a binary variable, i.e. each cell is either occupied or empty, we have

$$P(\bar{m}|x_{1:t}, z_{1:t}) = \frac{P(\bar{m}|z_t, x_t)P(z_t|x_t)P(\bar{m}|x_{1:t}, z_{1:t-1})}{P(\bar{m})P(z_t|x_{1:t}, z_{1:t-1})}. \tag{5.8}$$

Dividing Eq. (5.7) by Eq. (5.8) and knowing that $P(A) = 1 - P(A)$, we have

$$\frac{P(m|x_{1:t}, z_{1:t})}{1 - P(m|x_{1:t}, z_{1:t})} = \frac{P(m|z_t, x_t)}{1 - P(m|z_t, x_t)} \frac{1 - P(m)}{P(m)} \frac{P(m|x_{1:t}, z_{1:t-1})}{1 - P(m|x_{1:t}, z_{1:t-1})}, \tag{5.9}$$

or

$$O(m|x_{1:t}, z_{1:t}) = O(m|z_t, x_t)O(m)^{-1}O(m|x_{1:t}, z_{1:t-1}), \tag{5.10}$$

if we define odds operation $O(x) = \frac{P(x)}{1-P(x)}$, which means $O(A) = \frac{P(A)}{1-P(A)} = \frac{P(A)}{P(\bar{A})}$ and $O(A|B) = \frac{P(A|B)}{1-P(A|B)} = \frac{P(A|B)}{P(\bar{A}|B)}$.

Using a logarithmic form results in a more natural additive scale for representing odds:

$$log\ O(m|x_{1:t}, z_{1:t}) = log\ O(m|z_t, x_t) - log\ O(m) + log\ O(m|x_{1:t}, z_{1:t-1}). \tag{5.11}$$

Knowing that $P(x) = \frac{O(x)}{1+O(x)}$, the occupancy possibility can be recovered from Eq. (5.10) and be expressed as follows:

$$P(m|x_{1:t}, z_{1:t}) = \left[1 + \frac{1 - P(m|z_t, x_t)}{P(m|z_t, x_t)} \frac{P(m)}{1 - P(m)} \frac{1 - P(m|x_{1:t}, z_{1:t-1})}{P(m|x_{1:t}, z_{1:t-1})}\right]^{-1}. \quad (5.12)$$

Eq. (5.12) is on how to update the occupancy probability of a grid map. Whenever a new measurement at location x_t is received, the occupancy probability is updated using the current measurement and the map built from the recent measurements. Often, the prior probability of m and the initial belief of the map are both assumed to be 0.5, while unknown, free, and occupied status are indicated by -1, 0, and 1, respectively.

To facilitate the updating of each block in the grid, and in particular, the area covered by the robot sensors, Bresenham's algorithm [105] is used which is optimal in the number of cells visited to project a line in a grid. The input arguments are the two ends (given in the form of integer) of a laser beam. Its usefulness is evident for the case of a laser beam, which in any case, may cover several grid blocks from the robot to the point where an obstacle is detected. Since we know the coordinates of both ends of the laser beam, (one end being the coordinates of the robot plus sensor offsets and the other end which can be calculated using simple trigonometry), the task is to find all points in between which correspond to a grid on the map.

To see the effect of Bresenham's algorithm, a close view of a portion of an occupancy grid map produced by a laser range finder is shown in Fig. 5.2. All grids which are in the path of a laser beam are updated accordingly. The choice of which grid to update is determined by Bresenham's algorithm. As more laser scans are read and updated, the map will get clearer. The advantage of using Bresenham's algorithm is its speed, since no floating point calculations are needed.

5.1.3 Scan Matching for Pose Estimation

The accuracy of updating a metric map depends crucially on the alignment of the robot with the map. The assumption that poses are known during mapping typically is not realistic due to slippage and drift and the existence of odometry noise, especially in the lack of a global positioning system, active beacons, or predefined landmarks. Incremental scan matching is an efficient, robust (relative) localization algorithms. To obtain the best estimate of poses, the process repeats of the following steps (see Fig. 5.3) until all the laser scans are mapped or user exits:

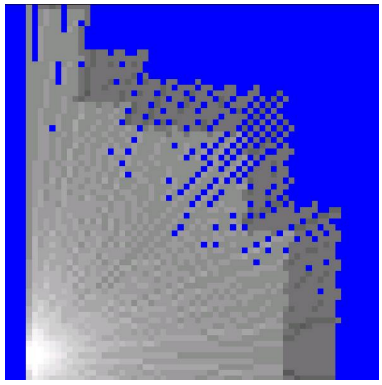


Figure 5.2: Effect of Bresenham's algorithm: a close view of a portion of an occupancy grid map produced by the laser rangefinder

(i) at the $(t-1)$ time instant, the robot is given an estimate of its pose and a global map;

(ii) take a new measurement (odometry or perception), and create a local map using N previous laser scans;

(iii) find the best fit for the latest laser scan in the local map. Take the new robot pose which gives the best probability of fitting the local map; and

(iv) update the global map with the latest scan and the estimated robot pose.

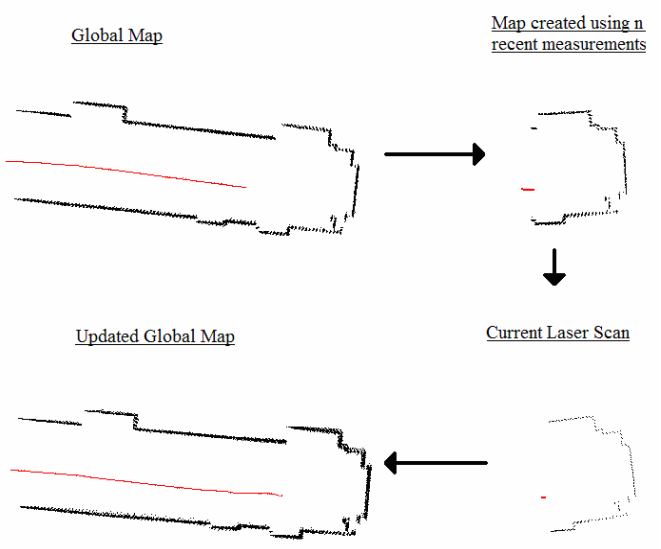


Figure 5.3: A single iteration of incremental scan matching algorithms.

In particular, incremental ML (maximizes likelihood) scan matching determines the most likely new pose such that the consistency of the measurement with the map and the consistency of the new pose with the control action and the previous pose are maximized. It makes a search in the space of all poses when a new (laser range) data item arrives, to maximize the likelihood of the t^{th} pose and map relative to the $(t-1)^{th}$ pose and map, given the motion model multiplied by the likelihood of the scan at that pose [52]:

$$\hat{x}_t = \text{argmax } \{P\left(z_t|x_t, \hat{m}(\hat{x}_{t-1}, z_{t-1})\right) \times P(x_t|u_{t-1}, x_{t-1})\}, \tag{5.13}$$

where the term $P(z_t|x_t, \hat{m}(\hat{x}_{t-1}, z_{t-1}))$ is the probability of the most recent measurement z_t given that the pose is x_t and the map built by far is $\hat{m}(\hat{x}_{t-1}, z_{t-1})$, and the term $P(x_t|u_{t-1}, x_{t-1})$ is the probability that the robot is at pose x_t given that the robot was previously at x_{t-1} and the last control command of motion is u_{t-1}.

With a stream of odometry $o_{1:t-1}$ and perception measurements $z_{1:t}$, the mapping problem can be phrased as recursive Bayesian Filtering for estimating the robot poses [106]:

$$P(x_{1:t}, m|o_{1:t-1}, z_{1:t}) = \alpha P(z_t|x_t, m) \int P(x_t|x_{t-1}, u_{t-1}) P(x_{1:t-1}, m|z_{1:t-1}, u_{0:t-2}) dx_{1:t-1}. \tag{5.14}$$

To reduce the complexity of this approach, the map built by far is instead constructed based on the most recent measurements only. There are two reasons for this, one being that the whole environment cannot be covered in a single scan, and the other one being that objects in the environment lead to occlusions so that many aspects of a given area are invisible from other positions. Therefore, measurements obtained from distant places often provide little information to maximize x_t.

Such a method does not keep tracking of any residual uncertainty. The advantage of this lies in its simplicity, which accounts for its popularity. The weakness is that, once the best pose is chosen, it will not be revised based on future data. The problem manifests itself in the inability to map big-scaled cyclic environments, where errors in the poses may grow without bounds. This is a general limitation of algorithms that do not consider uncertainties when building maps and that possess no mechanism to use future data to adjust past decisions.

5.2 Fusion of Laser and Sonar Data for Better Obstacle Detection

5.2.1 Motives of Fusing Laser and Sonar Data

When modeling a range sensor, we have assumed that a range reading indicates the presence of a target at that range. This is based on a diffuse reflection of energy from the target: if the surface roughness of the target is larger than the wavelength of the impinging beam, it will act like a point reflector, scattering energy equally in all directions. Specular reflection [107] of the wave occurs when the wave hits a surface at some angles or the surface of an object is smooth with respect to the wavelength of the beam. In this case, the reflected wave does not return to the sensor and thus the measured time-of-flight does not represent reflection from the nearest surface. This happens frequently for sonar especially in a confined environment. The consequence is that longer range readings will be resulted than the actual distances.

Sensor fusion involves the integration of data from various sensors into a uniform data structure. For the purpose of mapping, sensory information from various sensors need to be combined in order to get the best view of the surroundings of the robot. Compared to sonar, a laser rangefinder offers a more reliable system for measuring distances. Hence, it will be used as the primary device for robot mapping purposes. However, it is important to note the following:

(i) multiple sonar sensors provide redundancy and enables cross checking;

(ii) the properties of laser are such that a laser rangefinder cannot accurately sense materials like glass or mirrors, whereas the ultra sound wave sent by sonar can be easily reflected on glass;

(iii) commercial laser rangefinders are normally able to scan the surroundings two dimensionally only. In contrast, a sonar sensor detects obstacles in a cone-shaped range.

Therefore, a sensible solution to improve the quality of mapping, especially those used for navigation purpose, is to incorporate the various sensors available to the robot and to make full use of the advantages of various available sensors for robust map building and navigation.

5.2.2 Selective Method to Fuse Laser and Sonar Data

The first step of sensor fusion for a particular system is sensor modeling, which is to describe the sensor returns with a mathematical expression. Next, pre-processing of sensor data, such as filtering out noised sensor data, is necessary to ensure that data containing no or less erroneous ones are supplied to the sensor fusion process. Finally, a mathematical technique, like Bayesian algorithms or Kalman filtering, should be used to produce merged (fused) data, with uncertainties in data sources taken into account.

It is possible to use laser and sonar sensor readings at the same time to update the occupancy probability of a cell using Bayesian rule such as Eq. (5.12). However, laser and sonar sensors differ much in their sensor characteristics, especially on measurement accuracy and sensor noise. A grid map fused in this way is apparently not suitable for a scan matching algorithm to estimate robot poses. For this reason, two separate grid maps are maintained: one is m^l, the map built from only laser data, and the other is m^f, the map constructed by fusing laser and sonar data. After building the laser map m^l for pose estimation, the two types of sensor readings are integrated to build the fused map, which is to represent the environment for mapping or navigation purpose.

As specular reflections result in longer range readings than actual distances, Polaroid sonar sensors almost never underestimate the distance to an obstacle [108]. The proposed solution is to use laser range data as the primary mapping source, which is supplied to scan matching for pose estimation, while sonar data are used selectively to update the fused map m^f. The laser rangefinder is normally mounted facing the front. In this selective method, a sonar reading, say z_t^s, is used only when it is from one of the sonar sensors mounted in the front of the robot, and at the same time its value is shorter than the laser readings within the sonar cone α_s:

$$P(m_l^f|z_t^l, z_t^s, x_t) = \begin{cases} P(m_l^s|z_t^s, x_t), & \text{if } z_t^s < \min\limits_{i\in\alpha_s} z_t^l(i) \\ P(m_l^l|z_t^l, x_t), & \text{otherwise,} \end{cases} \tag{5.15}$$

where z_t^l represents the measured range readings ($z_t^l(i)$ is the i^{th} reading) from the laser rangefinder within the angle δ that covers the same area as the sonar cone, as shown in Fig. 5.4.

Fuzzy logic approaches are characteristic in that they deal with various situations without analytical modeling of the environment [25]. To facilitate autonomous

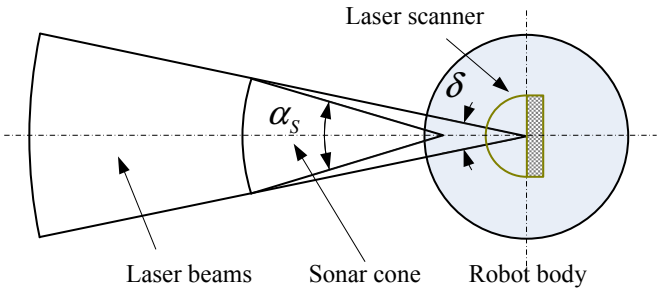


Figure 5.4: Selective use of sonar readings by comparison with corresponding laser readings.

map building, we implement a fuzzy controller for wall following similar to the one presented in Chapter 4.3.4. During navigation using this fuzzy controller for wall following, obstacle avoidance is achieved by constantly checking the area around the robot using both laser and sonar sensors. For the purpose of obstacle avoidance, sonar data have priority over laser range data because of their area of coverage — in the shape of a cone, which allows for better obstacle detection. In addition, sonar may detect some obstacles of materials that are difficult to be detected by a laser sensor. For collision avoidance purpose, only sonar range readings not too big (e.g. less than 2m) are taken, considering that sonar data with big range value are not very reliable and that reactive collision avoidance is determined by the immediate surrounding of the robot. If obstacles are detected, the fuzzy controller for wall following will make an appropriate change in the robot's moving direction.

5.3 Topological Map Creation

5.3.1 Motivation

Occupancy grid map is used to represent the (global) environment in this research most of time. Nevertheless, topological maps have certain advantages over occupancy grid maps: they take up less memory space and allow for various efficient path planning approaches. They are suitable to represent environments of bigger size or outdoor environments, where there are more chances to utilize landmarks for localization. As the size of a map increases, more resources must be allocated for map storage, and computation of searching for a path within the map. This may make occupancy grid map based navigation difficult to be carried out in real-time. On the

other hand, topological maps are not as detailed as occupancy grid maps. It would be good to be able to rely on both types of maps.

Therefore, it will be meaningful to convert an occupancy grid map into topological one after a significant portion of the environment has been mapped. We implement the process of converting an occupancy grid map into a topological map in a way similar to the methods mentioned in [109]. It involves the use of a thinning algorithm to reduce the unoccupied space in a grid map, and adding and chaining nodes using the skeleton to form the desired topological map.

In essence, topological map creation is normally an offline process which can be run after receiving a fixed number of laser range scans (e.g. 2500 range scans) or at the end of the whole process of constructing an occupancy grid map. The constructed grid map is then processed using techniques similar to image processing.

5.3.2 Skeletonization and Chaining Algorithms

The process of skeletonization involves the use of a thinning algorithm to reduce the unoccupied space in a grid map into a series of lines the width of one grid. This skeleton links together possible paths on which the robot may travel. This path is not optimized and merely leads to other unoccupied regions on the map.

Thinning algorithm is commonly used in image processing, e.g. Zhang-Suen thinning algorithm [110]. To reduce an occupancy grid map into the required skeleton, this research implemented a thinning algorithm with its pseudo code shown in Algorithm 7. For efficiency, a second round of thinning is exerted on the resulting nodes obtained from first iteration of thinning.

Algorithm 7 Thinning algorithm

Neighbors - $P_i, i = 2, 3, \cdots, 9$ are the defined as the 8 neighbors of a pixel P_1, or (x, y). The indices of them are counted in a CW manner, with the first one being P_2, or $(x-1, y)$.
Z01(P_1) - the number of zero to nonzero translations in the sequence $P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_2$.
NZ(P_1) - the number of nonzero neighbors of P_1.

1: Scan through all the points of the image.
2: Calculate Z01(P_1), NZ(P_1), for all points.
3: Delete P_1 if all of the following conditions are satisfied:
   i) $2 \leq \text{NZ}(P_1) \leq 6$,
   ii) $\text{Z01}(P_1) = 1$,
   iii) $P_2 * P_4 * P_8 = 0$,
   iv) $P_2 * P_4 * P_6 = 0$.

Fig. 5.5(b) illustrates the result of applying the thinning algorithm to the original shape (Fig. 5.5(a)), which is derived from an occupancy grid map that only contains

information of free space.

(a) Original shape derived from an occupancy grid map, with black region depicting unoccupied space.

(b) The shape is reduced into a series of lines.

Figure 5.5: An illustration of the thinning algorithm.

Before the chaining process is started, nodes must be found out first from the skeleton obtained by the thinning process. Nodes are defined as either end points or cross points. Each node stores the coordinates of its location and a link to any neighboring node. If only one neighbor of a pixel P_1 is free, i.e. $NZ(P_1) = 1$, this pixel is added to the topological map as an end node. If three or more neighbors of a pixel P_1 are free, i.e. $NZ(P_1) \geq 3$, and $Z01(P_1) \geq 2$, this pixel is added to the topological map as a cross node.

Next, the process of chaining tries to link all nodes to their neighboring nodes. The process of linking all nodes together is done recursively, which makes the code simple and efficient.

Fig. 5.6 illustrates the result of applying the chaining algorithm to the skeleton shown in Fig. 5.5(b).

5.3.3 An Example of Topological Map Creation

Creation of a topological map from the currently available occupancy grid map will be useful, since an occupancy grid map can be obtained in many scenarios. Fig. 5.7 shows the resultant topological map created from the occupancy grid map in Fig. 5.15. The process of skeletonization is used to reduce the unoccupied space into a

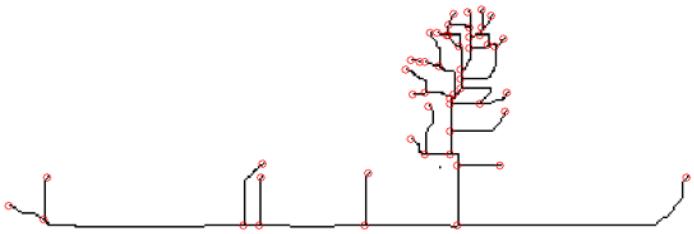


Figure 5.6: Nodes (depicted by small circles) are added to the skeleton by the chaining algorithm.

series of lines the width of one grid. Eventually, the topological map will consist of only nodes and edges. The nodes are defined as either end points or branches.

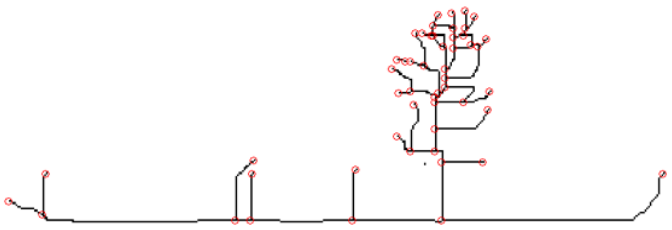


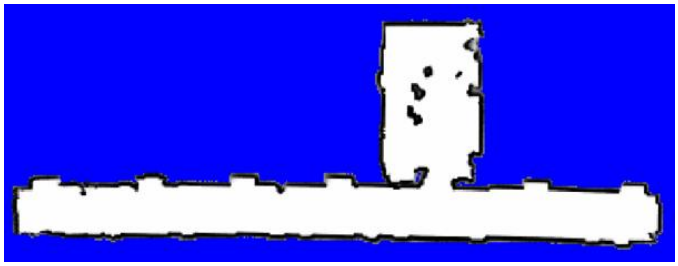
Figure 5.7: Resultant topological map created from an occupancy grid map.

Fig. 5.8(b) shows the resultant topological map created from a slightly modified version of the same occupancy grid map. It shows that there are fewer nodes in the room in the resultant topological map. This is attributed to the fact that the room is well defined in this case before creating the topological map.

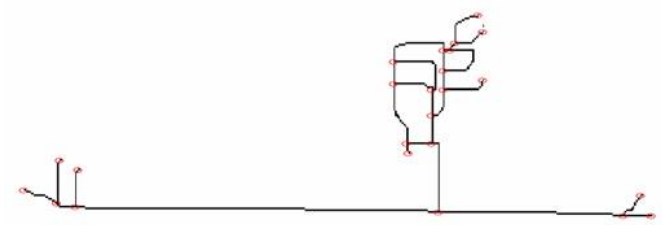
Comparing the two resultant topological maps, it can be seen that the thinning algorithm tends to produce more lines when the shape to be reduced has a lot of noise in it – in the form of numerous objects or unclear areas. This is a general limitation of the thinning algorithm. Due to this reason, the thinning algorithm may not work well for cluttered environments. A solution to this would be to ensure that an occupancy grid map is properly built and contains as little ambiguous areas as possible.

5.4 Simulations and Experiments

The online mapping program is implemented in C programming within the CARMEN architecture (see Appendix C). A desktop PC equipped with a Pentium III 900M CPU and 256M memory was used for computation (scan matching and mapping) and displaying. In experiments, the Magellan Pro robot was responsible for acquiring sensor data supplied to the PC and executing motion commands received.



(a) A slightly modified occupancy grid map with clearer objects in the room.

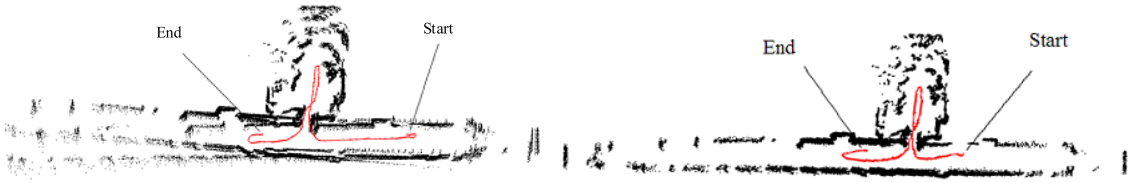


(b) Resultant topological map.

Figure 5.8: Another test of topological map creation.

The robot is with 16 sonar sensors (with a beam width of 30 degrees), and a laser rangefinder, SICK LMS 291 (see Table B.1 in Appendix B for more information such as its resolution).

5.4.1 Occupancy Grid Mapping and Scan Matching



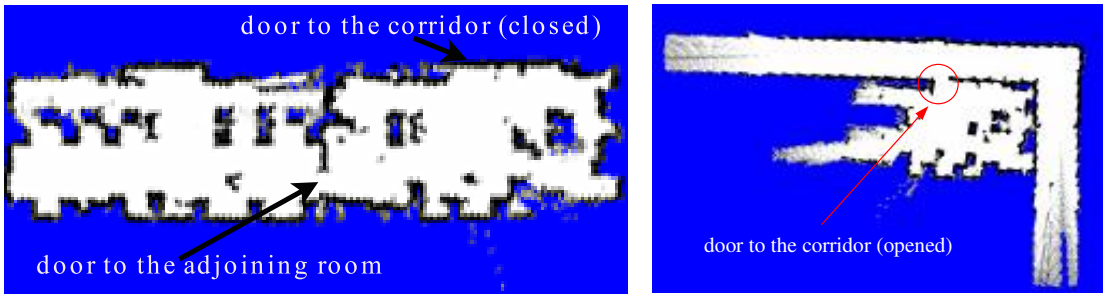
(a) Plots of odometry and laser scan raw data. (b) Result of simulation with scan matching.

Figure 5.9: Simulation of collecting laser data without/with scan matching used.

Fig. 5.9 shows the results of the simulation tests to plot laser scans when the robot was teleoperated. The robot started in a corridor, turned right into a room, made a turn and exited the room. The path, starting from the point labeled as "Start" and ending at the point labeled as "End", is denoted by solid lines. In Fig. 5.9(a), odometry (robot position) and laser scan data are plotted without using localization to correct the robot positions. A test (Fig. 5.9(b)) was run in a similar manner except

that scan matching was applied for pose estimation. This time, the mapping result looks more plausible as the scans are aligned correctly. The need for pose estimation such as scan matching is obvious, since laser scans of the corridor after the robot exited the room need to be aligned with previous scans.

Actual tests of the mapping and scan matching algorithms were run at the Student Projects Lab, National University of Singapore. It contains obstacle cluttered laboratory environments. As shown in Fig. 5.10, the results of mapping and scan matching at the Lab Room (E4A-06-11, NUS) and its nearby areas look good.



(a) Test at Lab Rom and its adjoining room. (b) Test at Lab Room and corridor.

Figure 5.10: Tests of scan matching and map building in laboratory environments.

Note that failing of the scan matching method is possible. This happened one time during about thirty experiment tests of mapping in non-large-scaled environments. As the robot entered the corridor from the Lab Room, it came into close proximity of 2 passers, by which caused the robot to misinterpret the laser range data, resulting in a misalignment of the outside corridor. The scan matching methodology is not robust enough to handle the dynamic change of environments (moving obstacles) when transiting from its current surrounding to a much different one.

A possible solution is to add a people tracker module which can stop the mapping process once a moving object is detected. Alternatively, a dynamic scan matching algorithm may have to be considered. An example would be DOGMA – dynamic occupancy grid mapping algorithm [111], which learns models of the dynamic environment.

5.4.2 Sensor Fusion of Laser and Sonar Data

Fig. 5.12 shows the resulting map obtained in a trial test by plotting both laser and sonar range data onto the same map. Fig. 5.12(a) plots the accumulated laser

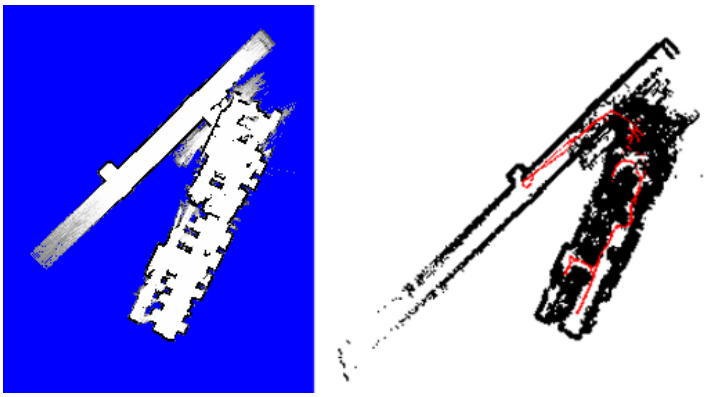
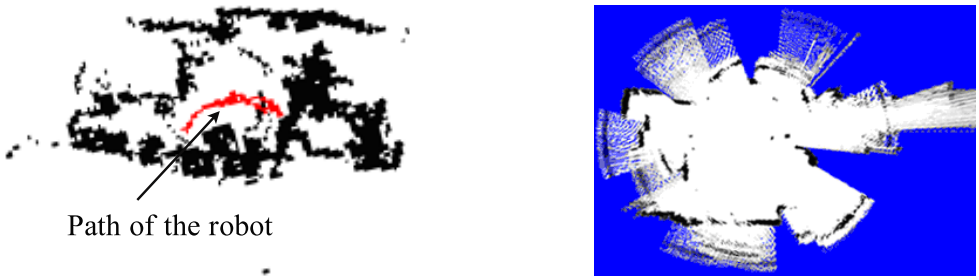


Figure 5.11: A failing of the scan matching method.

scans and the sonar readings as well as the robot trajectories. It can be observed that a simple combination of both laser and sonar range data does not necessarily produce a good map. As shown in Fig. 5.12(b), the inaccuracies of sonar measurements may distort the map constructed based on both sonar and laser data.



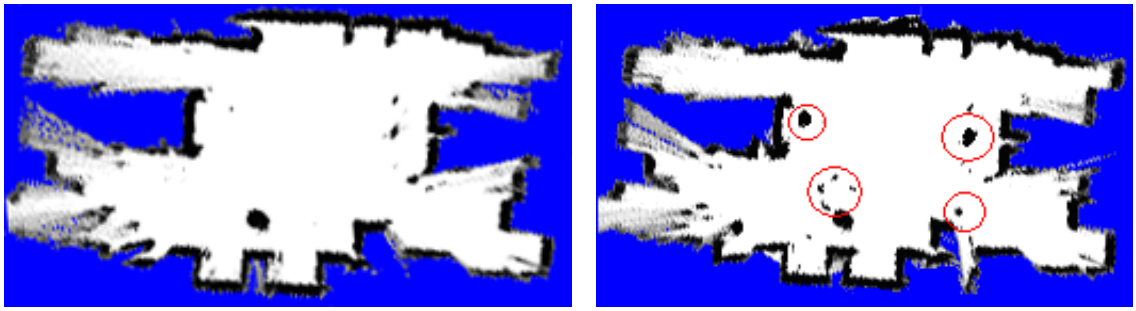
(a) Laser and sonar data.

(b) Map constructed.

Figure 5.12: Both laser and sonar range data are plotted onto the same map.

Another test used laser range data as the primary mapping source while sonar readings were used selectively using the method presented in Section 5.2.2. The resultant map in Fig. 5.13(b) compares favorably to the map in Fig. 5.13(a), which is constructed from laser data only. It is shown that the laser rangefinder is unable to detect all obstacles in the room out of its plane of vision, such as those objects indicated by small circles in Fig. 5.13(b).

Another test of the selective sensor fusion method is presented in Fig. 5.14 for further analysis. To help differentiate the two types of sensor data, purple patches are plotted to indicate the obstacles detected by sonar data. Though sonar data are usually inaccurate, they come in useful for the purposes of obstacle detection, and thus collision avoidance – allowing the robot to navigate safely without colliding into



(a) Map built with only laser range data. (b) Map constructed using laser and sonar data.

Figure 5.13: A comparison of maps built with laser data only and with the selective method of sensor fusion.

objects of height not at the same level of the laser rangefinder's view plan or hidden objects such as a glass door.

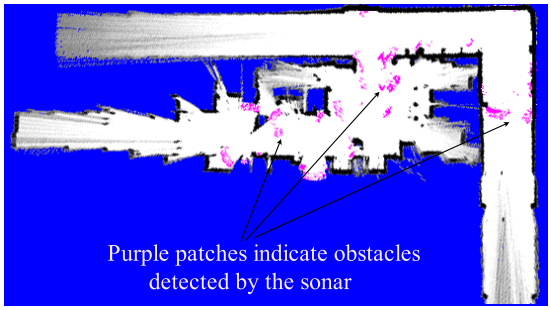


Figure 5.14: A test of sensor fusion at the Lab Room and the outside corridor.

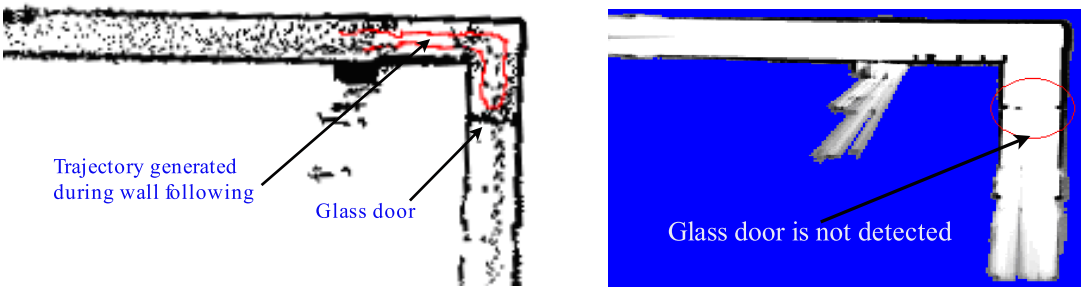
5.4.3 Sensor Fusion for Better Collision Avoidance

The fuzzy controller for wall following was tested through both simulation and experiment. The robot was ensured that it kept a distance of about 0.8m (from the center of the robot) to the wall on its right at all times while exploring the environment. Fig. 5.15 plots a simulation result of the wall following algorithm. It shows the path taken by the robot at the top part of the figure, and the resulting occupancy grid map at the bottom part.

Fig. 5.16(a) shows the result of the accumulated laser scans, and the accumulated sonar data as well as the robot trajectories obtained in an experiment of wall following in the corridor environment. A combination of both laser and sonar data were used in this experiment. At the end of the corridor is a closed glass door which the robot was able to detect with the use of sonar sensors. To illustrate this, the resulting occupancy

Figure 5.15: Laser and sonar data collected in a simulation test of wall following.

grid map created with laser data only is shown in Fig. 5.16(b). It is obvious that the laser rangefinder was unable to correctly detect the closed glass door. The use of sonar sensors allows the system to detect objects better, and thus improves the robustness of obstacle avoidance.



(a) Laser and sonar data, and robot trajectories.

(b) Map constructed during wall following using laser data only.

Figure 5.16: Laser and sonar readings, robot trajectories, and grid map obtained by an experimental test of wall following.

5.5 Summary

This chapter has presented a practical methodology for building occupancy grid maps online for autonomous mobile robots. The latest laser or sonar measurements are efficiently updated to the global grid map using Bayes' theorem. Incremental ML scan matching is applied for pose estimation, such that relative position localization can be done in permitted time for map building to be carried out online. The selective method makes use of the advantages of both laser and sonar sensors, and improves

both obstacle detection and mapping accuracy. The fuzzy controller for wall following allows the robot to build maps autonomously, and an offline method is implemented to convert the constructed occupancy grid map into a topological one for large maps and outdoor environment applications.

As verified by the simulation and experimental results, the proposed approaches have the following advantages: i) the selective method, by maintaining a laser map and a fused map, enables satisfactory pose estimation achieved based on available scan matching techniques; ii) fusion of laser and sonar data improves both obstacle detection and mapping accuracy, which helps to achieve better representation of the environment and robust collision avoidance that may not be achievable with 2D laser range readings alone; and iii) the fuzzy controller for wall following allows the robot to build occupancy grid maps autonomously with enhanced obstacle detection by using both laser and sonar data.

Chapter 6

Hierarchical Framework: Incremental Path Planning and Optimized Dynamic Motion Planning

This chapter studies a hierarchical approach for incrementally driving a mobile robot to its destination in unknown environments. A* algorithm is modified to handle a map containing unknown information, and based on it, optimal (discrete) paths are incrementally generated with a periodically updated map. Next, accelerations in varying velocities are taken into account in predicting the robot pose and the robot trajectory resulting from a motion command. Obstacle constraints are transformed to suitable velocity limits for the robot to perform relatively high-speed navigation while avoiding collisions when needed. Then, to trace the waypoints, the system searches for a waypoint-directed optimized motion in a reduced one-dimensional translation or rotation velocity space. Various situations of navigation are dealt with by using different strategies rather than a single objective function.

6.1 Incremental Dynamic Path Planning with Partial Map

The section presents an incremental search algorithm for replanning robot paths with a periodically updated map.

6.1.1 Modified A* Search for Partially Known Environments

A Star Search

A* search employs a "heuristic estimate" that ranks each node by an estimate of the best route that goes through that node. It visits the nodes in order of this heuristic estimate. In each step of a search, a weight function f is used to order the queue. Let a state correspond to a grid cell of the map, and define the following two cost functions:

- $g(s)$ - the actual cost of going from the initial state to the current state s; and
- $h(s)$ - a heuristic estimate of the cost of going from the current state s to the solution (goal) state.

The potential of reaching the goal via state s is then evaluated by the estimated total cost given by:

$$f(s) = g(s) + h(s), \tag{6.1}$$

where $g(s)$ can be taken as the length of the path linking the geometric centers of the traversed regions (cells), and $h(s)$ as the straight-line distance from state s to the goal position. Since $h(s)$ is an optimistic estimate which never overestimates the actual cost, it is then guaranteed that the first plan that is encountered is an optimal plan w.r.t. the real cost function $g(s)$ (though not necessarily optimal in the global sense).

An open list and a closed list are maintained to keep track of the progress of A* search. The open list is initialized with the start state as the first node, and the closed list is initialized to be empty. The following steps are repeated until the open list is empty:

1: The node with the lowest f-value in the open list is removed, and labeled as the *current* node. If the removed node is the goal node, A* search will exit and return a complete chain of nodes for the path to the goal.

2: A neighboring node s (in the eight directions) of the current node is added to the open list at a place according to its f-value, if it is not in the open list or it is matched to a node inside the open list which has a higher f-value.

3: If node s is matched to a node inside the closed list (which consists of nodes that have been checked) which has an equal or lower f-value, no further processing will be done on node s. Otherwise (i.e. if node s has lower cost than the stored node), the stored node is replaced by node s.

Dynamic Searching in Partially Known Environments

The robot may have only partial information about the environment before it begins its A* search, as *a priori* model maps are rarely available. D* algorithms are capable of planning paths in partially known, and changing environments. However, D* search is much harder to implement successfully because of the added dimensions characteristic to the problem of attempting to navigate with real-time replanning in partially known environments.

Similar to the work of [112], this research uses A* algorithm for search considering its robustness in finding an optimal path. To handle a map containing unknown information, the condition of finding a path is accordingly defined as follows [113]: if the removed node is the goal node, the successful path is reconstructed, and the algorithm ends; or if the removed node is unknown, a possible path is found, and the algorithm ends. In the latter case, no sufficient knowledge is available to ensure that there exists a feasible path planned between the second last node and the goal node.

6.1.2 Obstacle Enlarging and Addition of Obstacle Cost

In this research, occupancy grid map is used since it provides more detailed information about the environment (compared with geometric or topological map), and can be easily updated when there is a sensor input due to the probabilistic nature of grids. The planner first creates a configuration space from the grid map considering the robot dimensions: the grid map is processed by enlarging each occupied grid to its neighboring free grids by a value bigger than the size of the robot (or r_{rob}, the radius of a circular robot):

$$r_{\text{enlarge}} = c_{\text{enlarge}} \cdot r_{\text{rob}}, \tag{6.2}$$

where c_{enlarge} (larger than one, and typically in the range [1.2, 1.5]) is a coefficient about how conservative the safety margin should be to ensure the safety of the robot.

If the A* algorithm is applied onto such a configuration space, it can be found that some parts of the obtained optimal path may be located very close to obstacles.

"Obstacle cost" is thus assigned to each free cell based on d_{nearest}, its distance from its nearest obstacle cell(s) in the C-space constructed. Obstacle cost is defined as follows such that it will be infinitely big if the distance is within r_{enlarge}, zero if the distance reaches the obstacle influencing distance, $r_{\max}(> r_{\text{enlarge}})$, and proportional to the inverse of the distance otherwise:

$$c_{obs} = \begin{cases} \infty, & d_{\text{nearest}} < r_{\text{enlarge}} \\ \alpha_{obs}/d_{\text{nearest}}, & r_{\text{enlarge}} \leq d_{\text{nearest}} \leq r_{\max} \\ 0, & d_{\text{nearest}} > r_{\max}, \end{cases} \tag{6.3}$$

where α_{obs} is a pre-defined coefficient to determine the significance of obstacle cost. In this way, the planner is ensured not to generate waypoints that are too close to obstacles while still able to provide a path optimal or almost optimal.

During the search subsequently carried out, both occupied and unknown cells are treated as occupied ones, and each free grid node in the constructed C-space is assigned two values: a g-value and an h-value, with an addition of obstacle cost to $g(s)$, i.e.

$$g(s) = g_a(s) + c_{obs}(s), \tag{6.4}$$

where $g_a(s)$ is the cost of traveling a cell length or $\sqrt{2}$ fold of a cell length, depending on whether node s is directly or diagonally neighboring to the current node that is just removed from the open list as the root node.

6.1.3 Path Straightening and waypoint Generation

The discrete search (A* search algorithm or other path planners) may result in jagged across grids. Straight paths typically look more plausible than jagged paths, particularly through open spaces. Therefore, after the discrete path sequence is obtained, redundant points are removed and only a set of waypoints which are connected by straight line segments are left. Algorithm 8 describes such a method to further optimize the path generated by the search. The algorithm outputs a set of waypoints: $(x_i, y_i), i = 0, 1, \cdots$, where the first point is the start point of the robot and the last one is the goal point.

6.1.4 Incremental Planning Algorithm

Fig. 6.1 shows the flowchart of the proposed incremental planning algorithm. Initially, it sets up basic settings such as the robot's initial pose, the goal and the

Algorithm 8 waypoint Generation.

```
1: Choose the first and the last path nodes as the start and end node, respectively. The first start
       node is set as the new waypoint.
2: while waypoints cannot be reduced any more do
3:     Connect the start and end nodes with a straight line.
4:     if this line collides with obstacles then
5:         Choose the mid path node between the start and the end nodes as the new end node.
6:     else
7:         The end node is added in as a new waypoint and set as the start node.
8:         The last path node is set as the new end node.
9:     end if
10: end while
```

information (size and resolution) of the grid map[1]. After a partial map is created with laser and odometry sensory data, the planner is able to plan/replan a path with the following procedure:

1: **Search and Planning**

Based on the C-space and the obstacle cost information, an optimal path, in the form of a series of nodes, is planned from the current state to the goal using the modified A* search algorithm. As only a partial map is available, such a path can be either a path to the goal node, denoted by "Path_To_Goal", or a possible path to it (i.e. the path from the second last node to the goal node contains unknown areas), denoted by "Possible_Path".

2: **Moving and Sensing**

The robot moves toward its target node while collecting information about its surroundings. When the target node is reached (i.e. within a certain distance from the robot), the robot will decide its next action as follows:

i) If the target node is the goal, the path planning task is accomplished;

ii) If the target node is the second last node in the series of nodes and the search result is "Possible_Path", the procedure will go to Step 3;

iii) Otherwise, the robot will continue this step to approach the remaining nodes in the plan.

[1]Coordinate transformation is frequently involved when implementing the proposed incremental path planning algorithm. Appendix A describes about converting coordinates between different coordinate frames.

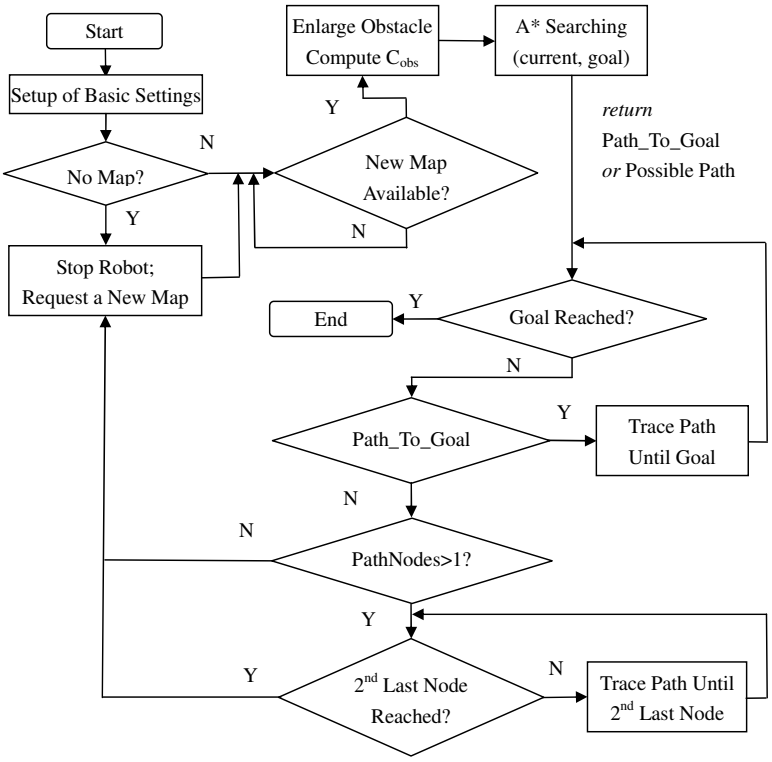


Figure 6.1: Flowchart of incremental search and planning algorithm.

3: **Re-localization for Next Round of Planning**

Before the robot is able to go to Step 1 to replan a new path, the robot sends requests for the best estimate of the relative position between it and the goal as well as the updated map information of the world.

Remark 6.1. *Because it takes a while to update/recieve the map supplied for search and to carry out such a graph search, we do not want to update this map while the robot is moving or else, it may run into an obstacle or deviate from the planned path. The robot is therefore commanded to stop, and the system subsequently replans a new path (a series of waypoints) to the goal. To the best of our knowledge, grid-map-based deliberate planning approaches seldom choose to continue driving the robot during that period unless there is a global path to guide the robot to continue moving forward. Nevertheless, the system's performance (mainly the average robot speed) could be enhanced if this restriction can be relaxed.*

6.2 Predicted Admissible Robot Trajectory under Robot Dynamics

This section presents admissible robot motion subject to various dynamic/actuator constraints and resulting robot trajectories that conforms to forward kinematics of a robot.

6.2.1 Forward Kinematics of Differential Drive Robots

We recall the kinematic modeling of differential drive robots that is presented in Chapter 2.2. As the reference point (RP) to trace a path is chosen at the midpoint of rear-axle, the robot's translation velocity v is defined at this point. Thus, we have

$$\dot{x} = v\cos\theta,\ \dot{y} = v\sin\theta,\ \dot{\theta} = \omega, \tag{6.5}$$

where θ denotes angle (of the main axis of the robot) w.r.t. the global frame, in this chapter.

Since the robot is frequently adjusted (accelerated or decelerated) during navigation, the velocities are normally time-varying in a control period. At a relatively short interval of Δt, we may take that the wheels undergo a fixed rate of acceleration or deceleration and thus we have

$$\begin{cases} v(t) = \frac{a_R + a_L}{2}t + \frac{v_{R0} + v_{L0}}{2} = at + v_0 \\ \omega(t) = \frac{a_R - a_L}{b}t + \frac{v_{R0} - v_{L0}}{b} = \varepsilon t + \omega_0, \end{cases} \tag{6.6}$$

where v_0 and ω_0 are the initial values of translation and rotation velocities, respectively.

With Eqs. (6.6) and (6.5) and the initial condition of integral as (x_0, y_0, θ_0), the solution for the robot pose is:

$$\begin{cases} x(t) = \int_0^t (at + v_0)\cos\left(\frac{1}{2}\varepsilon t^2 + \omega_0 t + \theta_0\right)\ dt + x_0 \\ y(t) = \int_0^t (at + v_0)\sin\left(\frac{1}{2}\varepsilon t^2 + \omega_0 t + \theta_0\right)\ dt + y_0 \\ \theta(t) = \frac{1}{2}\varepsilon t^2 + \omega_0 t + \theta_0, \end{cases} \tag{6.7}$$

where the position can be obtained only through numerical integration method such as Simpson's Rule.

6.2.2 Admissible Motions Satisfying Dynamic Constraints

Due to the actuators' limits, there exist maximum limits on the robot velocities and the robot accelerations. Let $a_{\text{s, max}}$ denote the maximum acceleration that the robot can be comfortably and smoothly accelerated, and $a_{\max}$ denote the maximum permissible deceleration that the robot can be slowed down and stopped without causing skidding to occur. If no reverse movement is allowed for the robot, to achieve a smooth and (actuator) feasible robot motion or simply called "admissible" motion, the following constraints will be imposed on the desired translational and rotational velocities (v_1, ω_1):

$$0 \le v_1 \le v_{\max}, v_0 - a_{\max}\Delta t \le v_1 \le v_0 + a_{\text{s, max}}\Delta t, \tag{6.8}$$

$$-\omega_{\max} \le \omega_1 \le \omega_{\max}, \omega_0 - \varepsilon_{\max}\Delta t \le \omega_1 \le \omega_0 + \varepsilon_{\max}\Delta t. \tag{6.9}$$

where $v_{\max}, \omega_{\max}$, and $\varepsilon_{\max}$ are the maximum translation velocity, maximum rotational velocity, and maximum rotational acceleration of the robot, respectively.

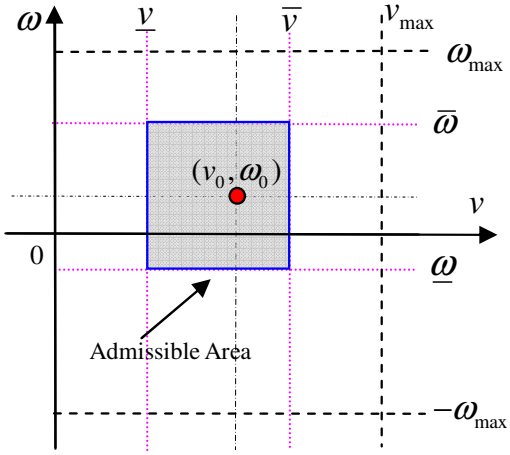


Figure 6.2: Region of admissible translation and rotation velocities.

As shown in Fig. 6.2, the admissible motion commands are confined by a rectangular region, i.e.

$$\mathbf{V}_a = \{(v_1, \omega_1) | \underline{v} \le v_1 \le \overline{v},\ \underline{\omega} \le \omega_1 \le \overline{\omega}\}, \tag{6.10}$$

where

$$\underline{v} = \max(0, v_0 - a_{\max}\Delta t),\ \underline{\omega} = \max(-\omega_{\max}, \omega_0 - \varepsilon_{\max}\Delta t),$$
$$\overline{v} = \min(v_{\max}, v_0 + a_{\text{s, max}}\Delta t), \overline{\omega} = \min(\omega_0 + \varepsilon_{\max}\Delta t, \omega_{\max}).$$

6.2.3 Trajectories Generated by Admissible Motion Commands

What would be the robot trajectory like within the duration of Δt if it is commanded to a certain motion, under the constraints (6.10)? Let us first consider the initial and final turning radii (the distance from the rotation center to the RP). If the desired target velocities are known as $v_1 \in [\underline{v}, \overline{v}]$ and $\omega_1 \in [\underline{\omega}, \overline{\omega}]$, the turning radius at a given time instant $t \in [0, \Delta t]$ can be evaluated using the following formula:

$$r(t) = \frac{v_0 + (v_1 - v_0)t/\Delta t}{\omega_0 + (\omega_1 - \omega_0)t/\Delta t}. \tag{6.11}$$

Apparently, the turning radius at the initial and final robot configurations will be given by

$$r_0 = v_0/\omega_0, r_1 = v_1/\omega_1. \tag{6.12}$$

Unfortunately, r_1 is often not known beforehand as normally the target velocities are to be decided.

Without losing generality, suppose that $\omega_0 > 0$ and $\overline{\omega} > 0$ (like the one illustrated in Fig. 6.2). Table 6.1 lists the special cases of possible final turning radii if (fixed) acceleration/deceleration is imposed on v and ω. Next, let us explore some useful properties about robot trajectories.

Table 6.1: Special Cases of Possible Robot Trajectories.

Final Turning Radius	Trajectory	Remarks
${}^1r_1 = \underline{v}/\overline{\omega}$	*traj* 1	max deceleration on v and max acceleration on ω applied.
${}^2r_1 = v_0/\omega_0$	*traj* 2	no acceleration.
${}^3r_1 = \overline{v}/\underline{\omega}$	*traj* 3	max acceleration on v and max deceleration on ω applied
${}^4r_1 = \underline{v}/\underline{\omega}$	*traj* 4	max deceleration v and max deceleration on ω applied.
${}^5r_1 = \overline{v}/\overline{\omega}$	*traj* 5	max acceleration on v and max acceleration on ω applied.

Property 6.1. *The value of* $r(t)$ *belongs to a certain range determined by* ${}^1r(t)$, ${}^3r(t)$ *and* ${}^4r(t)$.[2] *Specifically,*

$$\begin{cases} r(t) \in [{}^1r(t),\ {}^3r(t)], & \text{if } \underline{\omega} > 0 \\ r(t) \in [{}^1r(t), \infty] \bigcup [-\infty,\ {}^4r(t)], & \text{if } \underline{\omega} \leq 0. \end{cases} \tag{6.13}$$

[2]Prefix superscript of $r(t)$ or $s(t)$ denotes the numbering of turning radius or arc length.

Proof. The above can be proved by evaluating Eq. (6.11). □

Property 6.2. *Arc length $s(t)$ at any time from the robot's initial position is between ${}^1s(t)$ and ${}^3s(t)$, arc lengths of trajectories ${}^1r(t)$ and ${}^3r(t)$, respectively.*

Proof. Arc length $s(t)$ can be expressed by

$$\begin{aligned} s(t) &= \int_{\theta_0}^{\theta_t} r\, d\theta = \int_0^t r\, \tfrac{d\theta}{dt} dt = \int_0^t r\omega(t)\, dt = \int_0^t v(t)\, dt \\ &= \int_0^t v_0 + t(v_1 - v_0)/\Delta t\, dt = \tfrac{v_1 - v_0}{2\Delta t} t^2 + v_0 t, \end{aligned} \tag{6.14}$$

from which we know that the arc length is proportional to the final translation velocity (when other variables in the equation are fixed), and thus it can be proved that $s(t) \in [{}^1s(t),\ {}^3s(t)]$. □

To illustrate this property more clearly, Fig. 6.3 plots the trajectories of a robot[3] with $v_0 = 0.3$, $\Delta t = 0.5$. The left and right diagrams show the robot trajectories generated during the last $\frac{1}{60}$s for $\omega_0 = 1.4$ (and thus $\underline{\omega} \geq 0$) and for $\omega_0 = 0.2$ (and thus $\underline{\omega} < 0$), respectively. The trajectories are distinguished by different colors according to the values of final velocities. It is shown that the arc length of an admissible trajectory is directly related to the value of v_1: the larger v_1 is, the longer the trajectory is.

In Fig. 6.4 the trajectories generated during the last $\frac{1}{60}$s are now distinguished by different colors according to the values of ending rotation velocities. It shows that ω_1 determines the angular displacement of the robot (for same v_1).

6.3 Situation-dependent Motion Optimization in Reduced Velocity Space

6.3.1 Admissible Collision Avoidance Considering Accelerations

The robot position at the end of a certain control period can be computed through numerical integration as Eq. (6.7). As shown in Fig. 6.3 and 6.4, in most cases

[3]The robot's dynamic constraints are $v_{\max} = 0.8$ m/s, $\omega_{\max} = 2.0$ rad/s, $a_{\max} = 0.8$ m/s^2, $a_{\mathrm{s},\max} = 0.5$ m/s^2, and $\varepsilon_{\max} = 1.6$ rad/s^2.

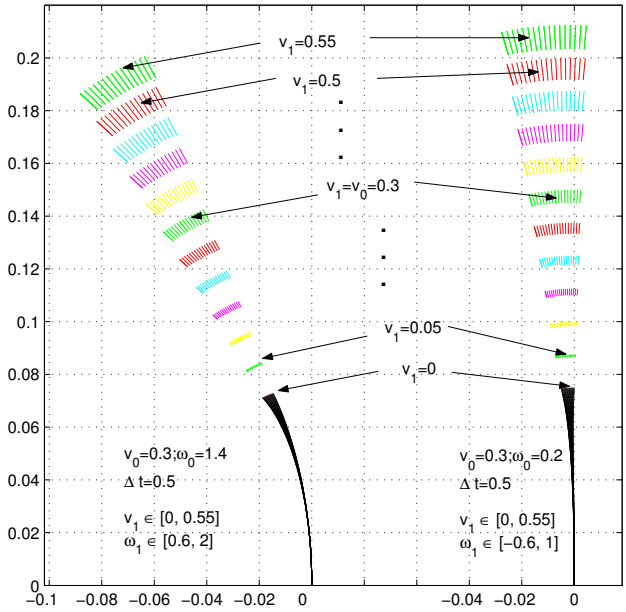


Figure 6.3: Trajectories generated during the last $\frac{1}{60}$second, and distinguished by different colors according to the value of ending translation velocities. (The entire trajectories for $v_1 = 0$ are plotted in black color.)

trajectories can be much different (especially when the duration is big) from the circular one, and thus may not be able to be approximated by a circle. This research takes into account the existence of accelerations, which is the general case, rather than assuming that the robot velocities remain constant in each control period and the robot path is thus a straight line or a circular path.

To obtain a satisfactory trajectory, first the commanded velocities should be admissible as discussed in the previous section. In addition, the selected motion command should result in a trajectory without intersecting with obstacles and at the meantime allow the robot to stop before it touches an obstacle, given the current robot position, and the actuator's deceleration capability of the robot. If the robot is commanded to (v_1, ω_1), the minimum time for the robot to be stopped is obtained by decelerating the robot with maximum acceleration without causing it to skid, i.e.

$$t_{\text{stop}} = v_1 / a_{\max}. \tag{6.15}$$

Similar to the proof of Property 6.2, the total arc length traveled by the robot, which moves for Δt and is subsequently commanded to stop, can be derived as follows:

$$\begin{aligned} s &= \int_{\theta_0}^{\theta_t} r \, d\theta = \int_0^{\Delta t} v(t) \, dt + \int_{\Delta t}^{\Delta t + t_{\text{stop}}} v(t) \, dt \\ &= \tfrac{v_0 + v_1}{2} \Delta t + \tfrac{v_1^2}{2 a_{\max}}, \end{aligned} \tag{6.16}$$

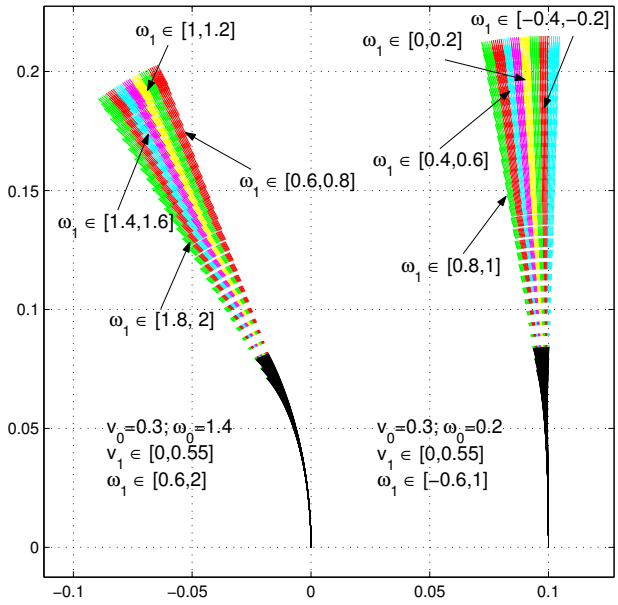


Figure 6.4: Trajectories generated during the last $\frac{1}{60}$second, and distinguished by different colors according to the values of ending rotation velocities.

which implies that the arc length of the extended trajectory for stopping the robot is proportional to the square of v_1.

Fig. 6.5 shows the entire trajectories generated if the robot moves at any admissible velocities for a duration of $\Delta t = 0.2$s and is subsequently commanded to stop. Let A, B, C, D denote the endpoints of the extended special trajectories (which include the extended part for the robot to stop) *traj* 1, *traj* 4, *traj* 5 and *traj* 3, respectively. For collision test purpose, the set of the robot trajectories obtained in a control period (0.2 s or less), could be approximated by a circular path, starting from the robot's current position and ending at a point denoted by E.

As shown in Figs. 6.5 and 6.6, the total area covered by the trajectories can be approximated by a region γ formed by points O, A, B, C, D, and E: the circular arc passing O and E, the sector $E\widehat{AB}$, and the sector $\widehat{AB}\widehat{CD}$. Of course, if $\underline{\omega} \geq 0$, $\widehat{AB}\widehat{CD}$ is not so accurate to represent the corresponding part of the trajectories, but still sufficient for predicting any potential collision.

The allowed travel distance $s_{\text{stop}}(O)$ is defined as the maximum possible arc distance for the robot to travel from the robot's current position O at any admissible velocity (under the current translation and rotation velocities and the robot dynamic constraints) for a duration of Δt, and subsequently to be commanded to stop without touching the obstacles (denoted as *obs*). It can be approximated as follows without

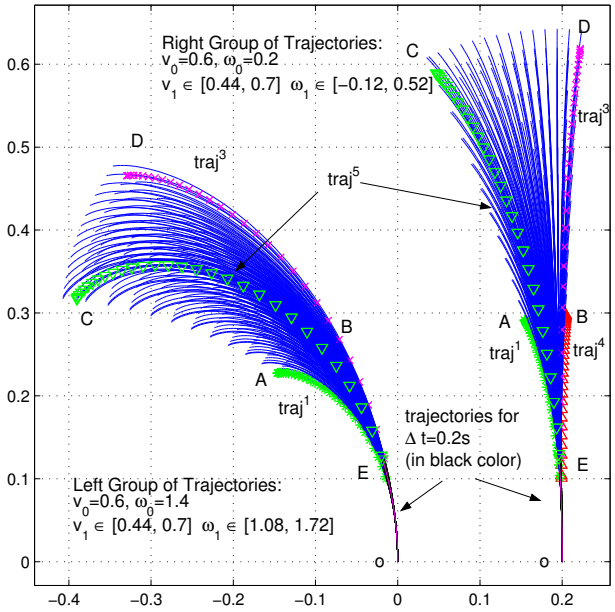


Figure 6.5: Trajectories (in black color) along which robot moves at admissible velocities for a duration of $\Delta t = 0.2$s, and trajectories (in blue color) that robot undergoes for it to be stopped subsequently.

being over-estimated:

$$s_{\text{stop}}(O) \approx \begin{cases} \infty, & \text{if } obs \bigcap \gamma = \emptyset \\ |\widehat{Oobs}|, & \text{else if } obs \bigcap \widehat{OE} \neq \emptyset \\ s_{\text{stop}}(O)_{E}^{\widehat{AB}} + |\widehat{OE}|, & \text{else if } obs \bigcap E\widehat{AB} \neq \emptyset \\ s_{\text{stop}}(O)_{\widehat{AB}}^{\widehat{CD}} + |\widehat{OE}| + |E, \widehat{AB}|, & \text{otherwise,} \end{cases} \tag{6.17}$$

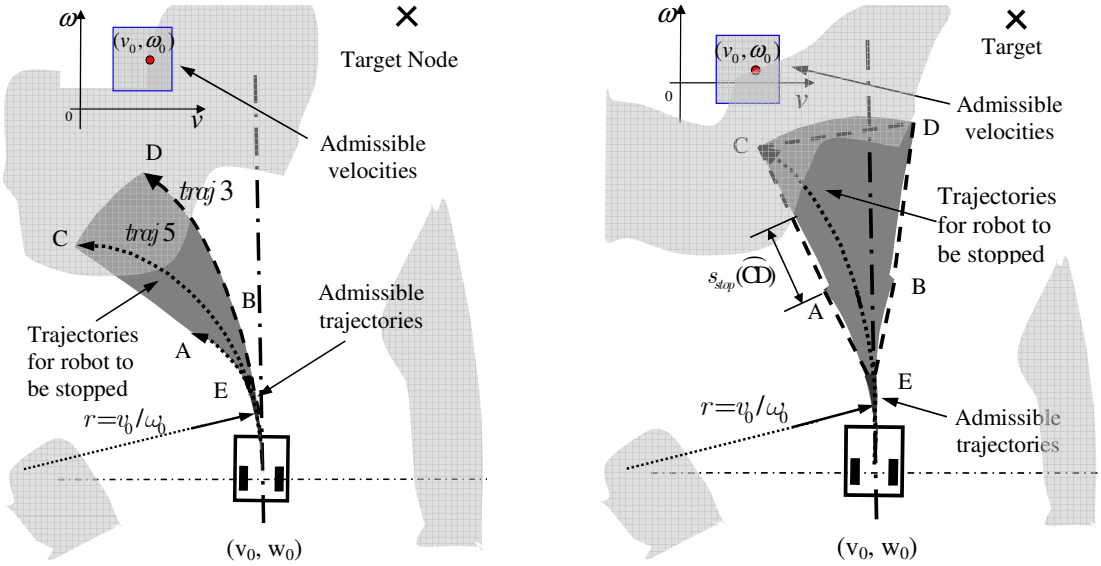
where $s_{\text{stop}}(O)_{E}^{\widehat{AB}}$ is part of the allowed travel distance $s_{\text{stop}}(O)$ counted from E to arc $\widehat{AB}$ while $s_{\text{stop}}(O)_{\widehat{AB}}^{\widehat{CD}}$ is part of $s_{\text{stop}}(O)$ counted from arc $\widehat{AB}$ to arc $\widehat{CD}$, and $|\widehat{Oobs}|$ is the arc (whose radius is approximated as v_0/ω_0) distance from O to the obstacle(s), respectively.

Then we are able to compare the value of the allowed travel distance $s_{\text{stop}}(O)$ with that of the furthest arc length $s(t_{\text{stop}})$ [as in Eq. (6.16)]. Collision-free motion can be ensured if the following constraint is satisfied:

$$s(t_{\text{stop}}) \leq s_{\text{stop}}(O), \text{i.e.} \frac{v_0 + v_1}{2}\Delta t + \frac{v_1^2}{2a_{\max}} \leq s_{\text{stop}}(O). \tag{6.18}$$

Given that $v_1, \Delta t$ and $a_{\max}$ are all nonnegative values, the limit of maximum translation velocity due to the obstacle constraints, v_{stop}, may be derived as follows:

$$\mathbf{V}_s = \{(v_1, \omega_1) | v_1 \leq v_{\text{stop}} = -\tfrac{a_{\max}\Delta t}{2} + \sqrt{a_{\max}(2s_{\text{stop}}(O) - v_0\Delta t) + \tfrac{a_{\max}^2 \Delta t^2}{4}}\ \}. \tag{6.19}$$



(a) $\underline{\omega} \geq 0$. (b) $\underline{\omega} < 0$

Figure 6.6: Illustration of allowed travel distance $s_{\text{stop}}(O)$ for the robot to safely stop without touching obstacles.

In comparison, Dynamic Window approaches consider velocities that do not hit an environment obstacle based on only the chosen velocity command and the assumption of circular robot motion:

$$\{(v_1, \omega_1) | v_1 \leq \sqrt{2dist(v_1, \omega_1) a_{\max}}, \omega_1 \leq \sqrt{2dist(v_1, \omega_1) \varepsilon_{\max}}\}, \tag{6.20}$$

where the function $dist(v_1, \omega_1)$ represents the distance to the closest obstacle on the curvature defined by the velocity pair (v_1, ω_1), measured by the product of the radius of the circular trajectory $r = v/\omega$ and the angle of the circular path that touches the nearest obstacle.

Combining Eqs. (6.10) and (6.19), the search space of admissible and collision-free velocities in this research will be

$$\mathbf{V}_1 \triangleq [\underline{v_1}, \overline{v_1}] \times [\underline{\omega_1}, \overline{\omega_1}] = \mathbf{V}_a \bigcap \mathbf{V}_s. \tag{6.21}$$

6.3.2 Motion Optimization in Event of Potential Collision

Provided that the robot has physical dimensions, the admissible region which covers all admissible trajectories is expanded by a radius defined as Eq. (6.2) before $s_{\text{stop}}(O)$ is computed. To reduce the computational load, we consider only a subset of

laser scans, which are of range within the maximum possible traveled distance that is determined by the current robot speed and the dynamics of the robot.

Situations of potential collision with obstacles and no potential collision are dealt with different strategies when searching for an optimized motion. In this chapter, *potential collision* is said to exist if and only if any laser scan of the subset falls within the expanded admissible region. The main task under this situation is to avoid collision with obstacles while approaching, if possible, the intermediate waypoint (or the target, denoted as g). The following are the optimization targets when searching for a desired motion command:

i) **Move to space with a bigger opening.** The movement of the robot is now expected to drive it to a place that is safe from collision with obstacles. In addition, alignment with the target or convergence to it is now of low priority.

ii) **Direct connectivity with the target.** To avoid being trapped in local minima, it is important to ensure a direct (straight-line) connectivity between the resulting robot position $\mathbf{q}_1(x_1, y_1)$ (corresponding to velocity (v_1, ω_1)) and the position of the target $\mathbf{q}_g$.

The requirement of Item (i) results in the robot favoring a rotation that drives it to a space within a subset of admissible angular velocities. Before searching for an optimized motion, the rotational velocity is randomly assigned a value within the favorable set of angular velocities which drive the robot to the space with bigger opening. This favorable set, denoted by $[\omega_\alpha, \omega_\beta]$, is determined when $s_{\text{stop}}(O)$ is computed.

Item (ii) is to ensure that the robot is able to reach each target, while it is noted that generally reactive planning methods (including direction and velocity space approaches) may get trapped in local minima. Collision test is carried out between the obstacle points and the rectangular ray expanded from the line segment $\overline{\mathbf{q}_1\mathbf{q}_g}$. A cost function about connection cost is defined as follows:

$$linkcost(\mathbf{q}_1, \mathbf{q}_g) = \begin{cases} 1, \text{ if } \mathcal{C}(\mathbf{q}_1, r_{\text{enlarge}}) \bigcap obs \neq \emptyset \\ \frac{1}{2}, \text{ if } \Upsilon_{\mathbf{q}_1}^{\mathbf{q}_g}(r_{\text{enlarge}}) \bigcap obs \neq \emptyset \\ 0, \text{ otherwise.} \end{cases} \tag{6.22}$$

The objective function to be evaluated in this situation is defined as follows:

$$f_{c,1} = c_v \frac{|\min(v_0, \overline{v_1})/2 - v_1|}{v_{\max}} + c_l linkcost(\mathbf{q}_1, \mathbf{q}_g), \tag{6.23}$$

where c_v and c_l are weights to adjust the importance of each item, and the first term in the right hand side of the equation indicates a preference for the robot to travel at around half of the minimum value between the maximum allowed speed and the current speed, i.e. $\min(v_0, \overline{v_1})$.

6.3.3 Motion Optimization in Absence of Potential Collision

The robot under this situation is expected to be commanded to approach the intermediate waypoint as fast as possible. Specifically, the following are the optimization targets when searching for a suitable robot motion:

i) **Better alignment with the target.** The alignment of the robot heading at $\mathbf{q}_1$ with the target should be relatively favorable compared with other motion candidates. It will be maximized if the robot faces directly to the target.

ii) **Decreasing distance to the target.** The distance of $\mathbf{q}_1$ from the target should decrease w.r.t. that of $\mathbf{q}_0$. This is to ensure that the robot converges to each waypoint given by the planner.

iii) **Direct connectivity with the target.** It will be favorable if there exists a direct (straight-line) connectivity between $\mathbf{q}_1$ and the target.

First the robot's new pose $\mathbf{q}_1$ is obtained through integration as in Eq. (6.7). Item (i) can be measured by the difference between θ_g^b and θ_g^{b1}, the target direction in the current robot body frame $\{b\}$, and that in the final robot body frame $\{b1\}$. In addition, the rotation should not be too large, as it accounts for localization errors (but this is of lower priority compared to the alignment to the target). A certain angular acceleration may occur when adjusting the angular velocity, and thus the orientation change, θ_1^b, should be computed as follows, rather than being evaluated by $\omega_1 \Delta t$ as is commonly done:

$$\theta_1^b = \theta_1 - \theta_0 = \frac{1}{2}\varepsilon \Delta t^2 + \omega_0 \Delta t = \frac{1}{2}(\omega_0 + \omega_1)\Delta t. \tag{6.24}$$

Item (ii) is evaluated directly based on the difference between the distances of the target from $\mathbf{q}_1$ and from $\mathbf{q}_0$. As before, Item (iii) can be evaluated via Eq. (6.22).

A minimization objective function is then designed as follows:

$$\begin{aligned} f_{c,2} &= c_g \frac{|\theta_g^{b1}| - |\theta_g^b|}{\pi} + c_d(|\mathbf{q}_1\mathbf{q}_g| - |\mathbf{q}_0\mathbf{q}_g|) + c_l linkcost(\mathbf{q}_1, \mathbf{q}_g) + c_\theta \frac{|\theta_1^b|}{\omega_{\max}} \\ &\triangleq f_{c,2}^g + f_{c,2}^d + f_{c,2}^l + f_{c,2}^\theta, \end{aligned} \tag{6.25}$$

where c_g, c_θ, c_d and c_l are weights to adjust the importance of each term.

The "stage" of the robot's current motion is determined according to the distances from the robot to the target waypoint and to the start waypoint ($\mathbf{q}_s$), and is categorized as follows:

$$stage = \begin{cases} S_{\text{Target(VeryClose)}}, & \text{if } |\mathbf{q}_0\mathbf{q}_g| < d_{\text{reached}} \\ S_{\text{Target(Close)}}, & \text{else if } |\mathbf{q}_0\mathbf{q}_g| \leq \max(d_{\text{close}}, v_0) \\ S_{\text{Start}}, & \text{else if } |\mathbf{q}_s\mathbf{q}_0| \leq d_{\text{closetostart}} \\ S_{\text{Middle}}, & \text{otherwise,} \end{cases}$$

where $d_{\text{reached}} < d_{\text{closetostart}} < d_{\text{close}}$ holds for the constants.

The parameter c_d is defined as follows such that decreasing distance to the target is preferred in the stage of S_{Middle}:

$$c_d = \begin{cases} c_d^*, & \text{if in } S_{\text{Middle}} \text{ and } |\mathbf{q}_1\mathbf{q}_g| > |\mathbf{q}_0\mathbf{q}_g| \\ 0, & \text{otherwise,} \end{cases} \tag{6.26}$$

where c_d^* [and c_θ^* in Eq. (6.27)] is a pre-defined constant. In addition, only motions satisfying $f_{c,2}^g + f_{c,2}^d < 1$ can be accepted in order to facilitate convergence to the goal.

The parameter c_θ is defined as follows, in view that bigger rotations are allowed when the robot is close to the start/target:

$$c_\theta = \begin{cases} c_\theta^*, & \text{if } \ stage = S_{\text{Middle}} \\ c_\theta^*/2, & \text{otherwise.} \end{cases} \tag{6.27}$$

6.3.4 Optimization Algorithm in Reduced Velocity Space

The objective (minimization) functions (6.23) and (6.25) are highly nonlinear. The solutions to them might be obtained only via numerical approaches. In this research, they are searched in the discretized set of velocity space. Algorithm 9 presents the procedure of obtaining a series of optimized motions for the robot to reach the specified waypoint. $Uni\_rand(min, max)$ is a uniform random function.

Unlike traditional velocity space approaches, translation velocity or rotation velocity is chosen before optimization is carried out. As shown in the procedure *choose_speed* (Algorithm 10), if no potential collision is detected, typically the robot favors a relatively low but accelerated speed when it moves from the start, a relatively low and decelerated speed when it is close to the target, and relatively high speed in the middle of moving to the target. In event of potential collision, as

Algorithm 9 MotionOptimization(waypoint $\mathbf{q}_g$)

while $\mathbf{q}_g$ is not reached **do**
  Update (v_0, ω_0) with odometry sensory data.
  wait until a new laser scan is ready.
  $\mathbf{V}_a \leftarrow (v_0, \omega_0)$ and robot dynamics. ▷ Eq. (6.10)
  $s_{\text{stop}}(O)$, favorite set $[\omega_\alpha, \omega_\beta] \leftarrow v_0, \omega_0$, $\mathbf{V}_a$, laser scan. ▷ Eq. (6.18)
  $v_{\text{stop}} \leftarrow s_{\text{stop}}(O)$; $\mathbf{V}_1 \leftarrow \mathbf{V}_a \bigcap \mathbf{V}_s$. ▷ Eqs. (6.19), (6.21)
  if potential collision detected **then**
    $f_{c,1}^{old} \leftarrow 2$; OptVelFound $\leftarrow$ FALSE.
    while OptVelFound=FALSE **do**
      $\omega_1 \leftarrow (\omega_\alpha + \omega_\beta)/2 + uni\_rand(\omega_\alpha, \omega_\beta)$.
      for each $v_1 \in$ "discretized set of" $[\underline{v_1}, \overline{v_1}]$ **do**
        if $f_{c,1} < f_{c,1}^{old}$ **then**
          $f_{c,1}^{old} \leftarrow f_{c,1}$; OptVelFound $\leftarrow$ TRUE; *break*. ▷ Eq. (6.23)
        end if
      end for
    end while
  else
    $f_{c,2}^{old} \leftarrow 4$; OptVelFound $\leftarrow$ FALSE; $searches \leftarrow 1$.
    while OptVelFound=FALSE **do**
      $v_1 \leftarrow choose\_speed(v_0, \underline{v_1}, \overline{v_1}, searches)$; $searches$++.
      for each $\omega_1 \in$ "discretized set of" $[\underline{\omega_1}, \overline{\omega_1}]$ **do**
        if $f_{c,2} < f_{c,2}^{old}$ and $f_{c,2}^{g} + f_{c,2}^{d} < 1$ **then**
          $f_{c,2}^{old} \leftarrow f_{c,2}$; OptVelFound $\leftarrow$ TRUE; *break*. ▷ Eq. (6.25)
        end if
      end for
    end while
  end if
  send velocity command (v_1, ω_1) to robot.
end while

Algorithm 10 choose_speed$(v_0, \underline{v_1}, \overline{v_1}, searches)$

Step 1: Adjust v_1 if necessary according to *Stage*:
S_{Start}: $v_1 \leftarrow v_0 + 0.5 \cdot a_{\text{s,max}} \cdot \Delta t$, if $v_0 < v_{\max}/4$.
S_{Middle}: $k \leftarrow 1$ if $v_0 < v_{\max}/2$, $k \leftarrow 0.5$ otherwise; $v_1 \leftarrow v_0 + k \cdot a_{\text{s,max}} \cdot \Delta t$.
$S_{\text{Target(VeryClose)}}/S_{\text{Target(Close)}}$:
if target is the goal or the 2n last waypoint **then**
  if $S_{\text{Target(VeryClose)}}$ **then**
    $v_1 \leftarrow v_{\min}$.
  else
    $v_1 \leftarrow v_0 - a_{\max} \cdot \Delta t$, if $v_0 > v_{\max}/2$.
    $v_1 \leftarrow v_0 - 0.5 \cdot a_{\max} \cdot \Delta t \cdot v_0/(v_{\max}/4)$, if $v_0 \in [v_{\max}/4, v_{\max}/2]$.
  end if
else
  $k \leftarrow 0.5$ if $v_0 > v_{\max}/2$, 0.25 if $v_0 \in [v_{\max}/4, v_{\max}/2]$, 0 otherwise.
  $k \leftarrow k \cdot 1.5$ if $S_{\text{Target(VeryClose)}}$. $v_1 \leftarrow v_0 - k \cdot a_{\text{s,max}} \cdot \Delta t$.
end if
Step 2: Obtain v_1 by adding some variation:
if $searches > 1$ **then**
  $v_1 \leftarrow v_1 + uni\_rand(-v_{\min}, v_{\min}) \cdot (v_1 < 2v_{\min}?0.5 : 1)$.
end if
$v_1 \leftarrow \underline{v_1}$ if $v_1 < \underline{v_1}$; $v_1 \leftarrow \overline{v_1}$ if $v_1 > \overline{v_1}$.

shown in Algorithm 9, rotation velocity is set as the median value of the favorable set $[\omega_\alpha, \omega_\beta]$.

Search in the two dimensional velocity space is thus reduced to one dimensional one. Of course, if, under the chosen translation/rotation velocity, no satisfactory rotation/translation velocity can be found, another translation/rotation velocity would be chosen for a new search.

In the process of generating a motion command, alignment to the target and convergence to it are considered with high priority, which are typically not taken into account by velocity space approaches. Waypoints in this research are designed in such a way that each pair of adjacent waypoints are visible to each other without being blocked by obstacles in the obstacle-expanded grid map. By ensuring connectivity and a decrease in the distance/angle to the target waypoint when possible, and applying the above velocity choice strategy with target-distance lookahead, the robot is able to reach a neighborhood area (radius $< d_{\text{reached}}$) of the goal or a second last waypoint (for the case of "Possible_Path"). For other kinds of waypoints, the robot need to reach a neighborhood area of bigger size in order to perform less deceleration in the robot speed. No investigation has been made on landing the robot at a target exactly considering it is beyond the research's main topics.

6.4 Simulation and Experimental Results

The proposed hierarchical framework was implemented in Linux/C programming language within the CARMEN architecture (Appendix C). In simulations and experiments, the same set of parameter values as shown below were used:

- Differential drive robot is circular and with radius 0.406 m.
- A laser rangefinder is mounted in the front of the robot. Its detectable range is 50 m, angular resolution 1°, and sampling rate 5 HZ.
- In simulations, errors (both range error and azimuth error) were introduced to the perceptions of the environment.
- Scan matching is used for the localization of robot poses.
- Computation and graphics display were performed on a Pentium IV PC (CPU 2.4G HZ and memory 1G byte) unless otherwise stated.

6.4.1 Reactive Point-To-Point Target Tracing

For comparison purpose, a reactive local motion planner is presented here which drives the robot from one waypoint to the next, which is provided by the same high-level path planner. A point-to-point motion command is generated[4] reactively as follows upon a receipt of perception sensory data:

$$(tran, rot) = \begin{cases} (0, rot_{\max}), & \text{if } |\theta_g^b| >= \theta_{\max} \\ (\min(tran_{\max}, |\mathbf{q}_0 - \mathbf{q}_g|), \theta_g^b), & \text{otherwise,} \end{cases} \tag{6.28}$$

where $tran_{\max}$ and $rot_{\max}$ are the maximum allowed translational distance and the maximum allowed rotational angle for a smooth motion, respectively, and $\theta_{\max}$ ($\leq rot_{\max}$) is a threshold angle – when the magnitude of θ_g^b exceeds it, the robot will be commanded to rotate only.

With the above strategy, the robot will try to track a straight line leading to the current target node from its current position, such that the distance traveled will be minimized unless the target waypoint's direction w.r.t. the robot is relatively big – in that case, the robot will be commanded to rotate (only) to face its front directly to the waypoint before further approaching it.

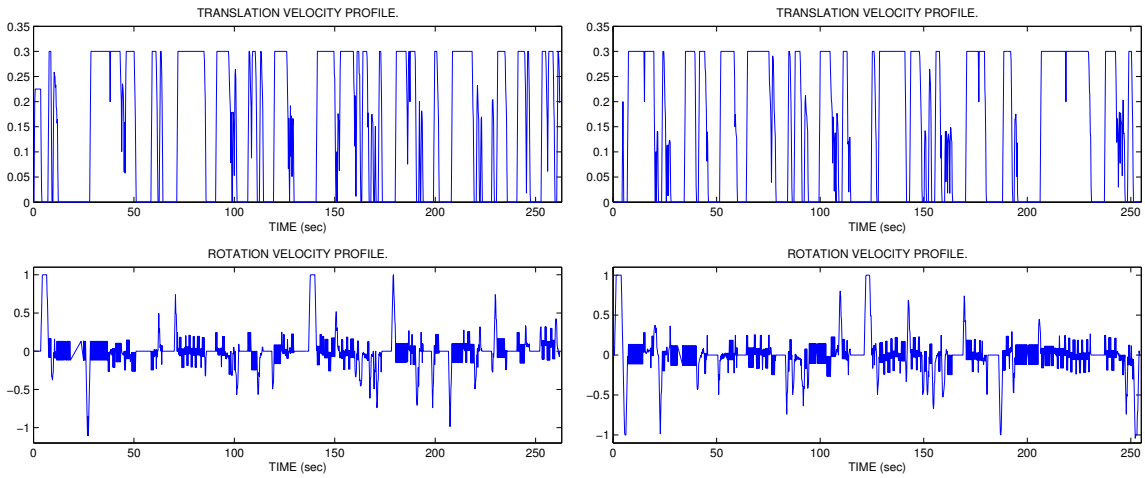
In the simulation and experimental tests presented in this subsection, additional parameters were set as follows:

- Maximum linear, and angular velocities of the robot are $v_{\max} = 0.30$ m/s, and $\omega_{\max} = 1.0$ rad/s, respectively. $tran_{\max} = 0.30$ m and $rot_{\max} = 0.20$ rad.
- Resolution of global grid maps is 0.025 m×0.025 m per cell, and size of the maps is 100 m × 100 m.
- Safety margin coefficient c_{enlarge} is set as 1.5 in simulations and 2.0 in experiments.

A simulation test was conducted where the position of the goal relative to the initial robot pose is (13.65, 14.83). Total time taken by the robot to reach the goal is 279.00 s, and the length of the entire path is 42.62 m. Another simulation test was conducted in the same environment. Total time taken by the robot to reach the goal, (-11.4, -20.18), is 311.23 s, and the length of the entire path is 44.97 m. The velocity profiles of the two simulation tests are depicted in Fig. 6.7(a) and Fig. 6.7(b), respectively. In the first simulation, the average speed (which has accounted

[4]which is subsequently converted to an optimized velocity command using a certain controller – this of course needs to be supported by the system.

for the periods when the robot was stopped during a search) is 0.153 m/s, about half of the max speed setting. In another simulation test, the average speed is 0.154 m/s, again about half of the max speed setting. It is also noted that sometimes the actual velocities became zero, during which the robot was stopped for replanning a new path.



(a) First simulation result.

(b) Second simulation result.

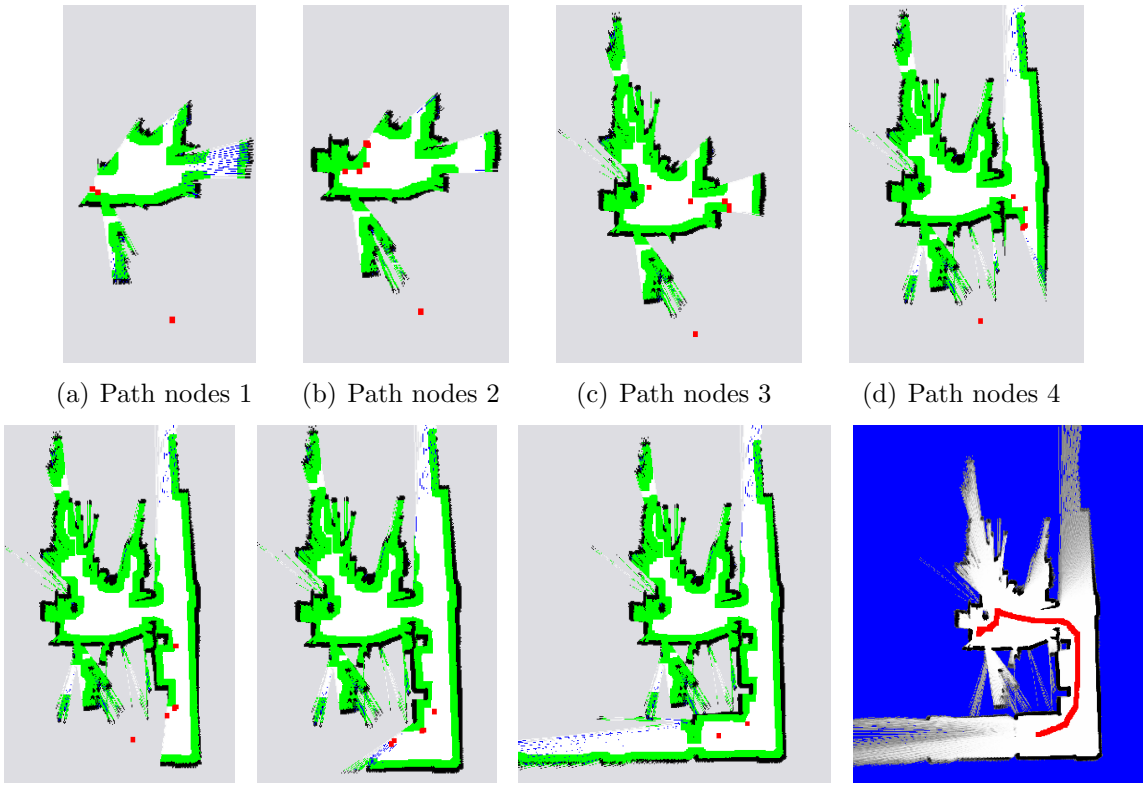
Figure 6.7: Profiles of translation and rotation velocities of two simulation tests of point-to-point target tracing.

The Magellan Pro robot (see Appendix B) was used for experiments. The desktop PC for computation (mapping and path planning) and displaying is equipped with a Pentium III 900M CPU and 256M memory. For the safety reason, the enlarging radius is set as 2 times of the robot's radius. Fig. 6.8 shows some of the scenes captured when the experiment was carried out.

The goal relative to the initial robot pose is approximately (3.6, 6.3). Fig. 6.9 shows the sequence of the robot trajectories (denoted by red circles of the robot size) right before each search and the path nodes or waypoints (denoted by small red squares) obtained subsequently by that search. Green color denotes the expanded part of the grid map that is supplied to search.

(a) Robot started from its initial position. (b) Corridor environment in experiment. (c) Robot stopped.

Figure 6.8: Snapshots of the experimental test. Robot was stopped by setting the original initial position as the target.



(a) Path nodes 1 (b) Path nodes 2 (c) Path nodes 3 (d) Path nodes 4

(e) Path nodes 5 (f) Path nodes 6 (g) Path nodes 7 (h) Final trajectories

Figure 6.9: Sequence of path nodes (denoted by small red squares) obtained in each search and final robot trajectories in the experimental test.

6.4.2 Simulation Results of Optimized Dynamic Motion Planning

Additional parameters were set as follows in simulations and experiments of optimized dynamic motion planning:

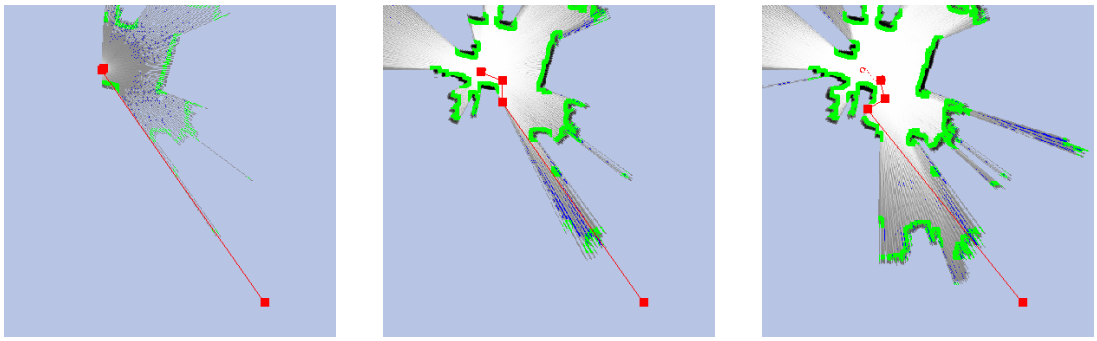
- Robot's dynamic constraints are $v_{\max} = 0.8$ or 1 m/s, $\omega_{\max} = 2.0$ rad/s, $a_{\max} = 1.2$ m/s^2, $a_{\mathrm{s,max}} = 0.5$ m/s^2, and $\varepsilon_{\max} = 2.0$ rad/s^2. In addition, $v_{\min}$ is set as 0.06 m/s.
- Resolution of global grid maps is 0.05 m×0.05 m per cell.
- Safety margin coefficient c_{enlarge} is set as 1.3.
- Optimization parameters are set as $c_g = 1$, $c_\theta^* = 0.2$, $c_d^* = 10$, $c_v = 1$, and $c_l = 1$.
- $d_{\mathrm{reached}} = 0.1$ m, $d_{\mathrm{close}} = 0.3$ m, $d_{\mathrm{closetostart}} = 0.15$ m.

Normal Stop Setting

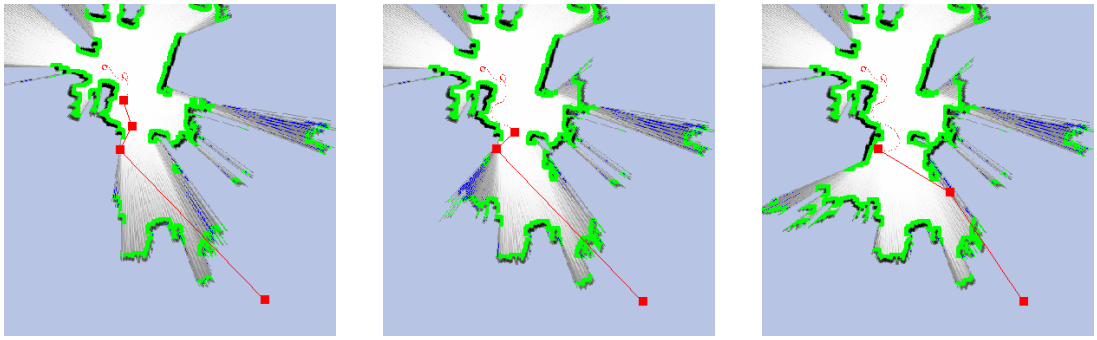
In the first simulation test (of the optimized motion planning approach), the whole process of navigation to the goal took the robot 12 times of search. Fig. 6.10 shows the sequence of incremental search. Each diagram plots grid map (supplied to the search), path nodes (denoted by small squares in red, and wired to their adjacent nodes), and robot trajectories.

Fig. 6.11 shows the laser measurements and the final robot trajectories of this test, where each robot pose is plotted as a circle of the robot size in red. Laser scans were continuously added to the plot without erasing previous ones along with the progress of navigation. To reach the goal, the robot took 83.44 s and traveled for a distance of 37.76 m.

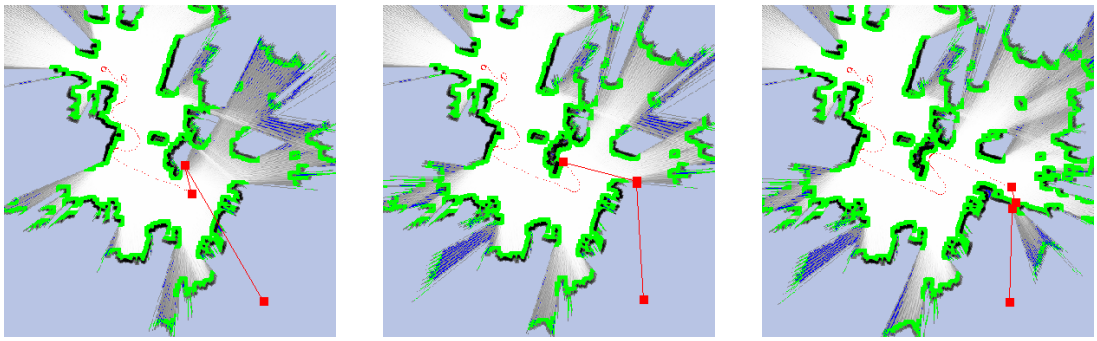
The velocity profiles obtained in this simulation are depicted in Fig. 6.12. It can be seen that the translation and rotation velocities were frequently adjusted, which was to speed up the robot when it was clear of obstacles or to avoid (potential) collision with obstacles. The average speed (which has accounted for the periods when the robot was stopped during a search) is 0.487 m/s. In addition, it can be seen that both the translation and rotation velocities are relatively smooth except that some vibrations of relatively small magnitude occur at some places. It is also noted that, sometimes, the actual velocities became zero, during which the robot was stopped for replanning a new path.



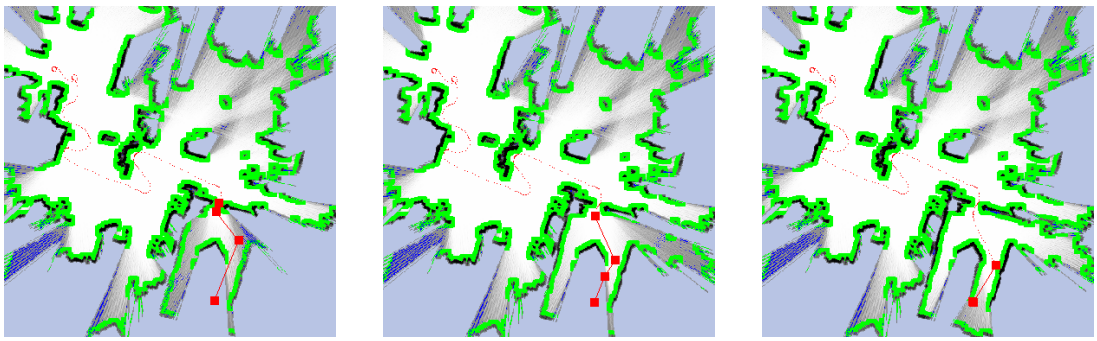
(a) Result of 1st search. (b) Result of 2nd search. (c) Result of 3rd search.



(d) Result of 4th search. (e) Result of 5th search. (f) Result of 6th search.



(g) Result of 7th search. (h) Result of 8th search. (i) Result of 9th search.



(j) Result of 10th search. (k) Result of 11th search. (l) Result of 12th search.

Figure 6.10: Path nodes obtained by each search in first simulation test.

Figure 6.11: Laser scans and robot's final trajectories in the first simulation (robot navigated from top left to bottom right). The position of the goal relative to the initial robot pose is (11.28, -16.14), or they are of straight-line distance 19.69 m.

Another simulation test was conducted in the same environment. The goal was located at (10.27, -22.00) relative to the robot's initial pose, or the straight-line distance between the start and the goal is 24.28 m. Fig. 6.13(a) shows the laser scans and the final robot trajectories (circles of the robot size in red). Fig. 6.13(b) plots the map supplied to the last search as well as the robot trajectories as a series of dots (the robot can be taken as a point in the C-space). The total time taken by the robot to reach the destination is 79.80 s, and the length of the entire path is 37.53 m. This result is similar to that of the first simulation test.

The velocity profiles of this simulation test are depicted in Fig. 6.14. The average speed is 0.469 m/s.

Slow Stop Setting

In order to achieve a smoother stop when needed, we attempted to use $a_{\mathrm{s,max}}$ instead of $a_{\max}$ in Eqs. (6.15) and (6.19) to compute t_{stop} and $s_{\mathrm{stop}}(O)$. In addition, the maximum speed is set as $v_{\max} = 0.8$ in this subsection. The third simulation test was carried out in the same environment as that of the first simulation test, while the fourth simulation test in a corridor environment. The velocity profiles of the two simulation tests are depicted in Fig. 6.15(a) and Fig. 6.15(b), respectively. In the third simulation, the average speed is 0.355 m/s, while the average speed is 0.370 m/s in the fourth simulation.

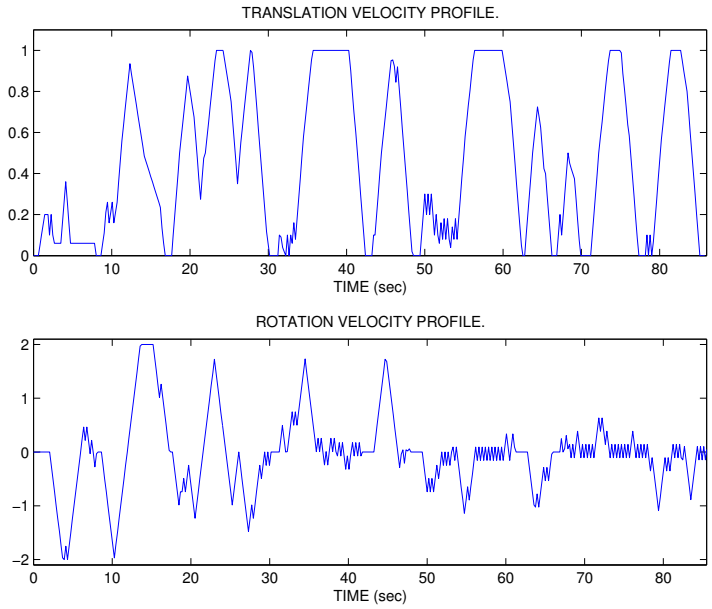
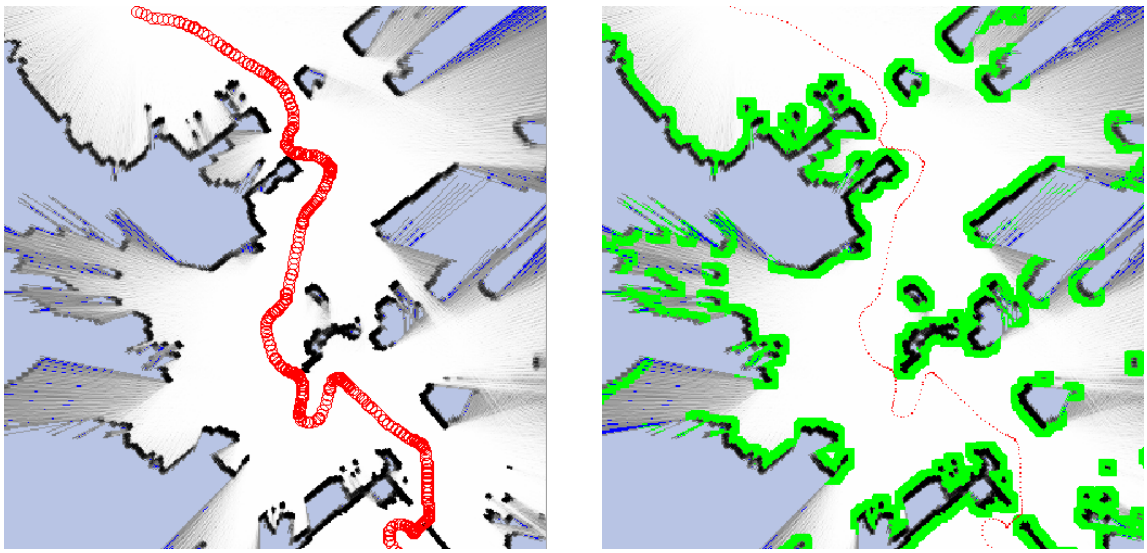


Figure 6.12: Profiles of translation and rotation velocities of the first simulation test.

Fig. 6.16(a) shows the laser scans and the final robot trajectories obtained by the fourth simulation test. The goal was located at (-10.69, -9.22) relative to the robot's initial pose, or a straight-line distance of 14.11 m. The total time taken by the robot to reach the goal is 244.94 s, and the length of the entire path is 86.96 m. Compared with the third simulation test, it took the robot much more time to reach the goal that is closer to the robot's initial pose. The explanation of this could be that at some key turning points, the partial map led the robot to decide a path that is not so optimal in the global sense. In a structured environment like the one in this test, the robot tends to travel a unnecessarily longer distance before it is able to "realize" it.

6.4.3 Experimental Results of Optimized Dynamic Motion Planning

The Magellan Pro robot was used for experiments. The deceleration limit $a_{\max}$ was set to be 1.0 m/s^2 instead of 1.2 m/s^2, because because it was observed that, when moving at its maximum speed limit, the robot may easily become unstable if it is exerted a too big deceleration (close to or around 1.2 m/s^2). Experiments were carried out in an unknown, unstructured laboratory environment as shown in Fig. 6.17.



(a) Occupancy grid map and robot trajectories. (b) Map supplied to search.

Figure 6.13: Robot's final trajectories and final grid maps of second simulation test.

In the first experiment, the goal relative to the initial robot pose was set to be (3.5 -5.2). The size of grid maps is 30 m $\times$ 30 m. Fig. 6.18 shows the robot trajectories (denoted by solid lines) right before each search and the path nodes (denoted by squares) obtained subsequently by that search. It is shown that the hierarchical algorithm is able to lead the robot to get closer to the goal incrementally. In addition, appropriate, optimized motions can be produced to drive the robot to its intermediate target, which might be located close to obstacles.

At the same time of online map building and displaying of the current laser scan and the robot trajectories, a video was taken during the experimental test. Fig. 6.19 shows the video captures when the robot was to search a path or when the robot was stopped.

The velocity profiles of this test are depicted in Fig. 6.26. The average speed is about 0.289 m/s. The velocity profiles of another test, which was carried out in the same environment, are depicted in Fig. 6.20. The average speed is about 0.337 m/s, a little bit higher than that of the previous experiment. The translation and rotation velocities are smoother than those obtained in the simulations. This is attributed to the fact that a robot in reality is not always able to rapidly track the motion commands, that is, the actual robot velocities will change more smoothly than the commanded ones as long as the change in the commanded ones are not too big. No trajectory controller other than a PID controller has been used in this research for

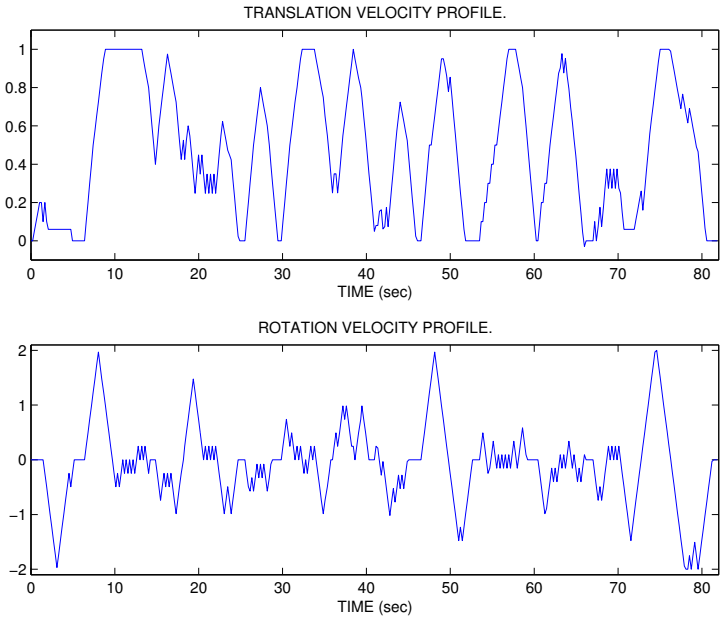


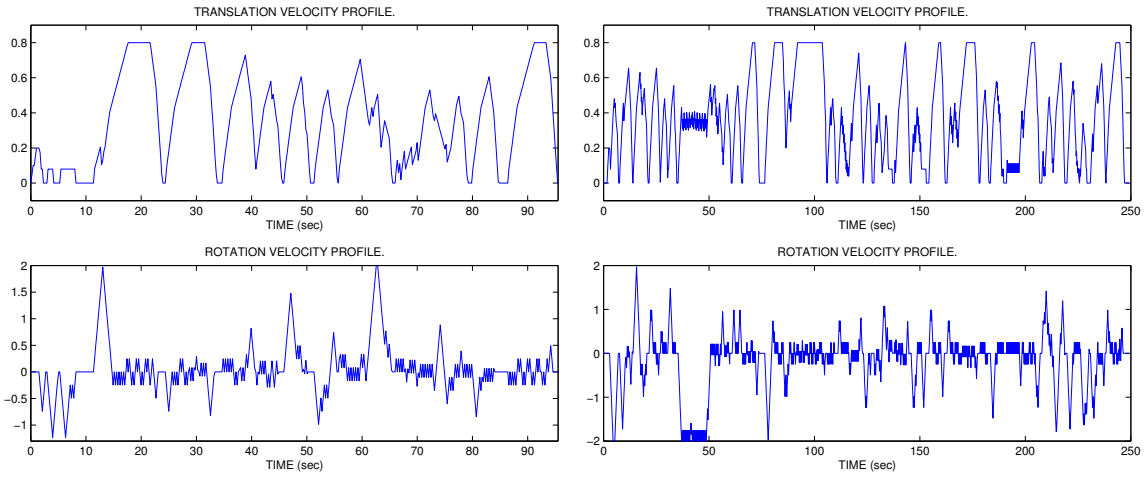
Figure 6.14: Profiles of translation and rotation velocities of second simulation test.

the robot to track the commanded velocity. On the other hand, the average speeds are slightly less than those indicated in the simulation tests.

In another experiment, the goal relative to the initial robot pose is approximately (3.2 -5.2). Fig. 6.21 shows the sequence of the robot trajectories (denoted by solid lines) right before each search and the path nodes (denoted by squares) obtained subsequently by that search. The velocity profiles of this test are depicted in Fig. 6.21(g). The average speed is significantly less than those indicated in the previous experiments. This is because the velocity commands were sent to the robot only upon a receipt of laser scan, i.e. about every 0.2 s, rather than upon a receipt of base message, i.e. about every 0.1 s –which is the right way to command the robot to the desired one (see Appendix B.2 for more explanations).

At the same time of online map building and displaying of the current laser scan and the travel robot trajectories, a video was taken during this experimental test. Fig. 6.22 shows the video captures when the robot was to search a path or when the robot was stopped.

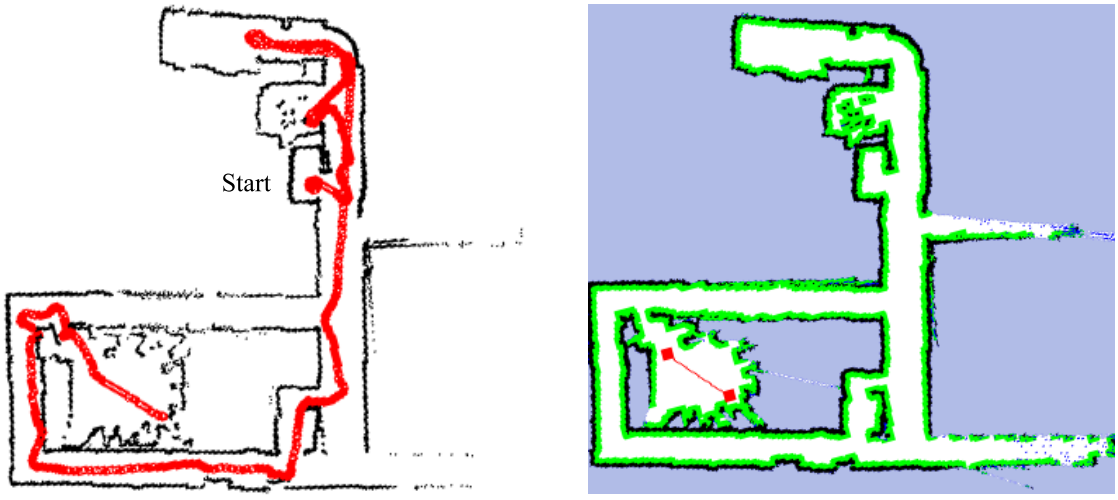
6.5 Discussions and Comparisons



(a) Third simulation result.

(b) Fourth simulation result.

Figure 6.15: Velocity profiles of the third and fourth simulation tests.



(a) Laser scans and robot's trajectories.

(b) Path nodes returned by the last search.

Figure 6.16: Robot's final trajectories, and grid map for A* search in the fourth simulation test (robot navigated from top to bottom).

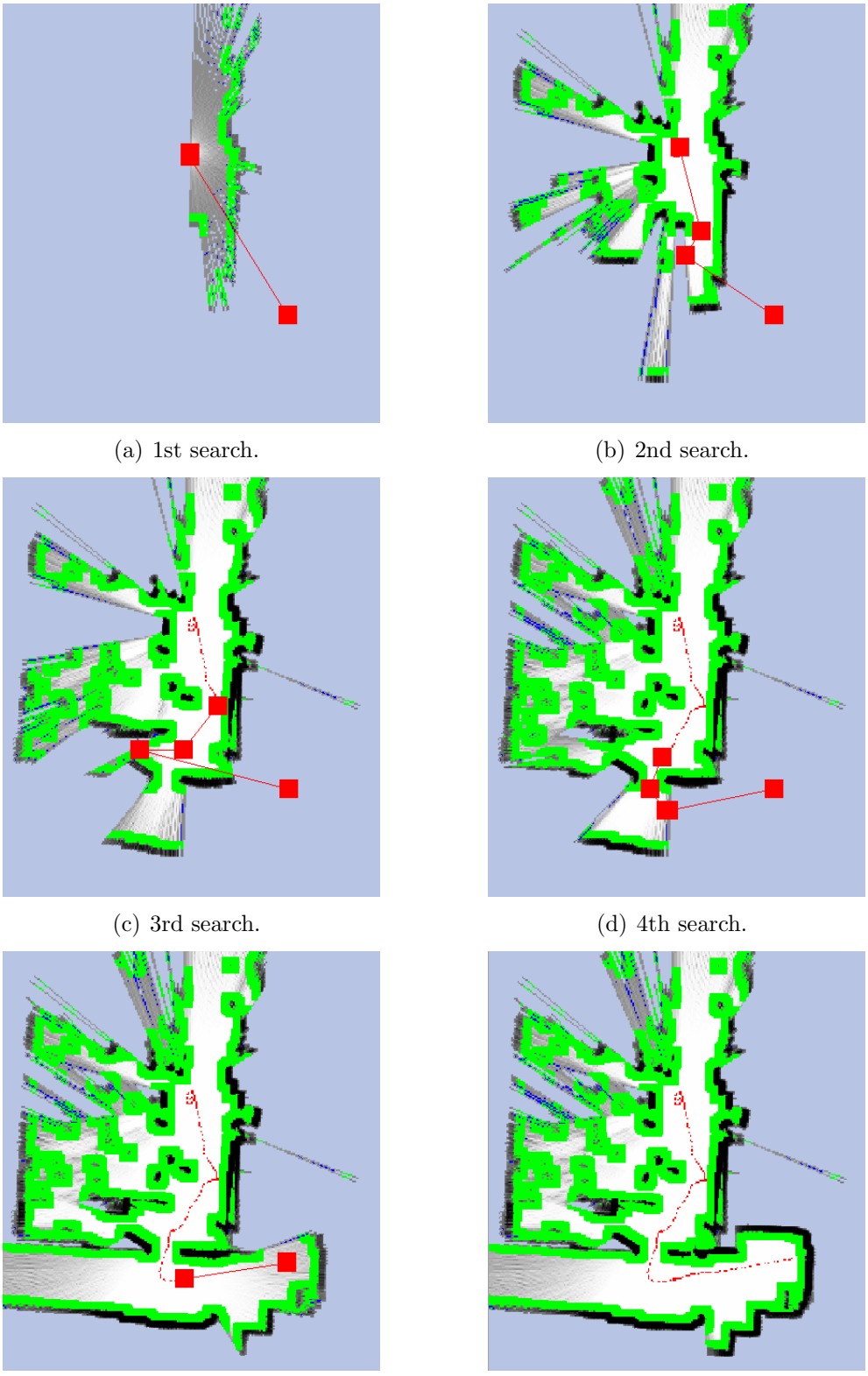
Figure 6.17: Magellan pro robot and laboratory environment for experiments.

6.5.1 Performance of Incremental Search

Table 6.2 presents the time used by each search in the four simulation tests, where start nodes are excluded when counting the number of path nodes (waypoints). It shows that a search of a map of scale 1600×1600 normally takes hundreds of ms or less and occasionally (1 out of 11 or 1 out of 25 here) takes about 2 or 3 seconds (note that localization, mapping and displaying modules cost much computational resources). Table 6.3 presents the time used by each search in the three experimental tests. The scales of grid maps are all 30 m × 30 m. Tens of other simulation tests suggested that the time spent to find a solution is of the same level.

Fig. 6.23 plots a statistics of time used vs. different map scales. Generally, the average value and the maximum value of search time increase with map size. However, it is not a proportional relationship, because search is performed on free space only (the size of which increases with each search) instead of the entire map. It is shown that a search of such a map of scale 800×800 or 1200×1200 (60 m × 60 m) takes average time of about 0.32 s (or less) and maximum time of about 0.96 s (or less). This is acceptable for most of indoor applications considering that simultaneous mapping and path planning is involved. When the map scale is 30 m × 30 m, the search time drops to 0.053 s (average value) or 0.28 s (maximum value).

Note that display of three maps (grid map to represent the environment, map for incremental search, and accumulated laser scans) at the same time costs much computation power and time, considering that drawing/displaying is done every 0.2 s



(a) 1st search. (b) 2nd search.

(c) 3rd search. (d) 4th search.

(e) 5th search. (f) Final robot trajectories.

Figure 6.18: Sequence of path nodes obtained in each search and robot trajectories in the first experiment.

(a) Searching for 1st path.

(b) Searching for 2nd path.

(c) Searching for 3rd path.

(d) Searching for 4th path.

(e) Searching for 5th path.

(f) Robot was stopped.

Figure 6.19: Snapshots of the first experiment when the robot was to search a path or when the robot was stopped.

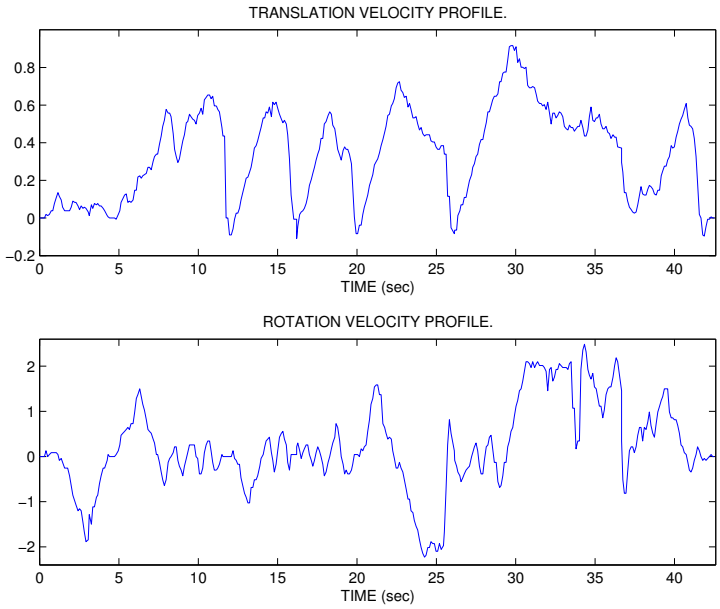
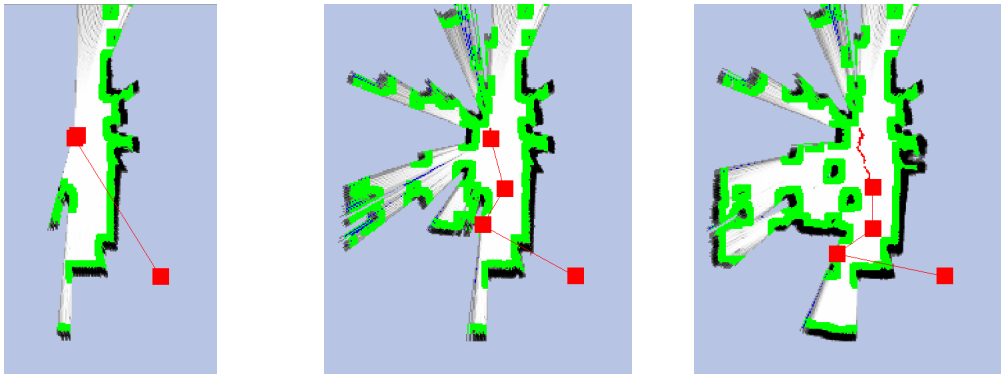


Figure 6.20: Profiles of translation and rotation velocities of the second experiment.

for as many as 1200×1200 pixels (for instance). If not for illustration purpose, display of the three maps can be disabled in order to enhance the system's performance (such as reducing the time used by the planner). In addition, for indoor applications, the detectable range of the laser scanner can be set as a smaller value (e.g. 20 m) instead of 50 m, to further reduce the time/computation used by the map updating process upon receiving of a new laser scan. It would be beneficial to apply these two measures, although no obvious timing or computational issues have been observed on the system throughout the simulation and experimental tests.

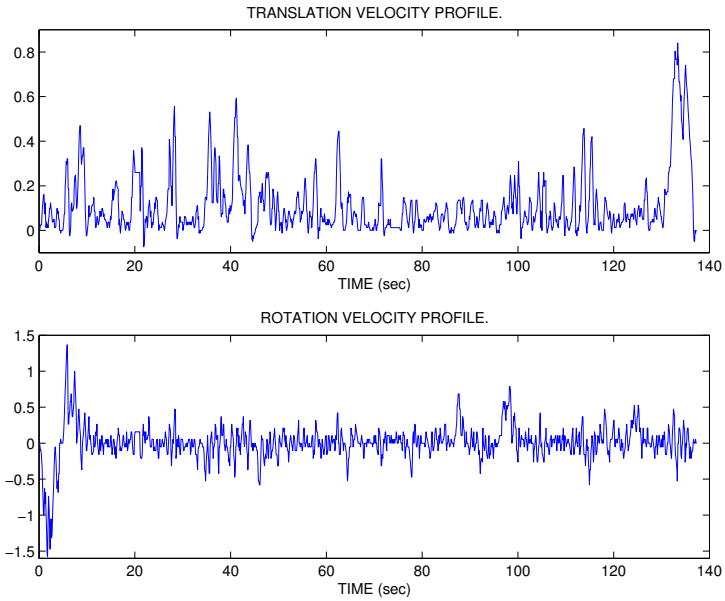
6.5.2 Robot's Average Speed

Factors such as actuator capabilities could significantly influence the average speed that a robot may achieve. Fig. 6.24 shows the average robot speeds grouped by different dynamic settings. In the "simulation (slow stop)" group, $a_{\mathrm{s,max}}$ was used instead of $a_{\max}$ in Eqs. (6.15) and (6.19) to achieve a smoother and elegant stop. As a result, the average speed dropped to 0.355 m/s or 0.370 m/s, compared to 0.469 m/s or 0.487 m/s in the "simulation (normal stop)" group. This is more obvious in experiments – the average speed could drop to a third of that the values in the "experiment" group. This observation can be explained by the fact that the robot dynamics and the forward kinematics have been taken into account to compute the



(a) Path nodes 1. (b) Path nodes 2. (c) Path nodes 3.

(d) Path nodes 4. (e) Path nodes 5. (f) Robot trajectories.

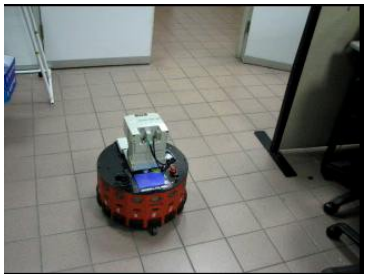


(g) Profiles of translation and rotation velocities.

Figure 6.21: Sequence of path nodes obtained in each search and robot trajectories and velocity profiles of the third experiment.

(a) Searching for 2nd path.

(b) Searching for 3rd path.



(c) Searching for 4th path.

(d) Searching for 5th path.

(e) Robot drove to goal.

(f) Robot was stopped.

Figure 6.22: Some snapshots of the third experiment when the robot was to search a path or when the robot was stopped.

value of $s_{\text{stop}}(O)$, which in turn determines the value of maximum translation velocity due to the obstacle constraints as in Eq. (6.19). A decrease in $a_{\max}$ (and thus t_{stop}) may cause a significant reduction in the allowed travel distance $s_{\text{stop}}(O)$, and the robot thus need to accelerate or decelerate much more frequently even when the obstacles are faraway.

The surroundings of the robot also have much influence on the average robot speed. It, together with the maximum deceleration $a_{\max}$, determines the value of $s_{\text{stop}}(O)$. Since the experimental environment is relatively obstacle-cluttered, the velocities were adjusted more frequently as more often there is a need to avoid (potential) collisions with obstacles. This explains why the robot's average speed in the "experiment" group (where $a_{\max}$ is set to be 1.0 m/s^2) is about 0.289 m/s or 0.337 m/s, and is a bit lower than that of the simulated robot. This also explains why only a small reduction is found in the average speed obtained by the simulation tests (less obstacle-cluttered environments) in which $a_{\text{s,max}}$ is used instead of $a_{\max}$.

Besides, the maximum speed setting has some impact on the possible average speed. Nevertheless, a decrease of the maximum speed from 1 m/s to 0.8 m/s alone may not result in a 20% decrease in the average speed. This can be easily seen from the robot's translation velocity profile – only for a small section of the whole time is

Table 6.2: Time Used by Search and Number of Path Nodes in Simulation Tests of Optimized Motion Planning.

	First Test		Second Test		Third Test		Fourth Test		Fourth Test(cont.)	
Index of Search	No. of Nodes	Time Used (s)	No. of Nodes	Time Used (s)	No. of Nodes	Time Used (s)	No. of Nodes	Time Used (s)	No. of Nodes	Time Used (s)
1 (14)	2	<0.01	2	<0.01	2	<0.01	2	<0.01	5	0.43
2 (15)	3	0.05	3	0.96	3	2.26	3	0.02	3	0.06
3 (16)	3	0.43	2	0.24	2	0.09	2	0.29	3	0.25
4 (17)	3	0.31	3	0.50	3	0.45	3	0.66	3	0.21
5 (18)	2	0.13	3	0.31	3	0.15	3	0.24	3	0.66
6 (19)	2	0.33	3	0.30	3	0.12	3	0.23	3	0.08
7 (20)	2	0.41	3	0.28	3	0.08	4	0.48	2	0.14
8 (21)	4	0.69	2	0.18	4	0.15	3	0.30	4	0.28
9 (22)	3	0.19	3	0.32	8	0.40	3	0.59	4	0.16
10 (23)	3	0.05	3	0.37	3	0.18	4	3.03	4	0.07
11 (24)	3	0.23	3	0.06	2	0.13	3	0.08	3	0.10
12 (25)	2	0.02					3	0.59	2	0.12
13							3	0.28		
Map Size	1200 × 1200		800 × 800		2000 × 2000		1600 × 1600			

Table 6.3: Time Used by Search and Number of Path Nodes in Experimental Tests of Optimized Motion Planning.

	First Test		Second Test		Third Test	
Index of Search	No. of Nodes	Time Used (s)	No. of Nodes	Time Used (s)	No. of Nodes	Time Used (s)
1	2	<0.01	2	<0.01	2	<0.01
2	3	0.04	3	0.02	2	0.04
3	3	0.05	3	0.09	3	0.11
4	4	0.02	3	0.01	4	0.06
5	2	0.02	3	0.09	3	0.28
6			2	0.01		
Map Size	600 × 600		600 × 600		600 × 600	

the robot able to reach the maximum speed.

It is noted that the maximum speed setting (1 m/s, determined based on the actuator capability of the Magellan Pro robot) and the average speed in our tests could be sub-par to that of state-of-the-art, though they are already high for a commonly used indoor mobile robot. The performance of the robot is, however, not necessarily sub-par, considering that the evaluation should instead be judged based on average robot speed, smoothness of motion, robustness in collision avoidance, convergence to the goal, optimality of path, among others, under the same conditions such as the test environment, and the availability of *a priori* map or a global path. Even when the robot speed alone is evaluated, the nature of our methodology should allow the robot to perform collision-free navigation at a higher average speed, under a higher

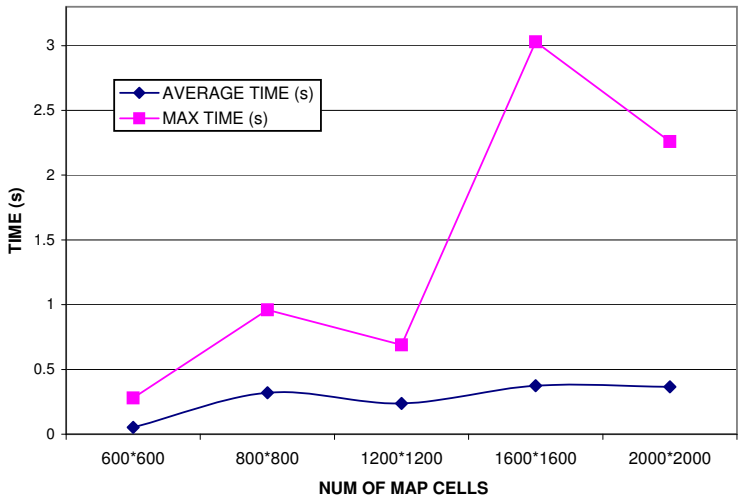


Figure 6.23: Statistics of time used vs. different map scales. The "600×600" group includes all the three experimental results, as they use grid maps of the same size.

maximum speed setting, or if a global path to the goal is provided (i.e. the robot is not required to stop in order to find a path to the goal).

6.5.3 Collision Avoidance When Very Close to Obstacles

The waypoints provided by the high-level planner are not always far from obstacles, though obtained in a way not too close to obstacles. As a consequence, in some instances, the robot needs to manoeuvre in obstacle-cluttered surroundings when approaching some waypoints. Figs. 6.25 (a)-(d) present some snapshots (with relative time stamp) of the first experiment before the robot was stopped for the third search [Fig. 6.25 (e)]. The robot was approaching a waypoint very close to the cabinets. Figs. 6.25 (f)-(g) show the laser measurements, the robot's location, and the motion direction, where both the robot dimensions and the obstacles are plotted in the same scale. The robot trajectories when the robot was stopped are plotted in Fig. 6.25 (h).

The corresponding translation velocity profile (Fig. 6.26) indicates that the robot started to decelerate from a speed of around 0.75 m/s at the time of about 3 seconds (see Fig. 6.25(a)) before the third search, which is located around 17.5 s in the velocity profile. From then on, the speed was observed to decrease rapidly (probably at the robot's maximum deceleration capability). Finally the robot was able to successfully stop itself with a small distance from the cabinets, as can be seen from the robot trajectories plotted in Fig. 6.25(h).

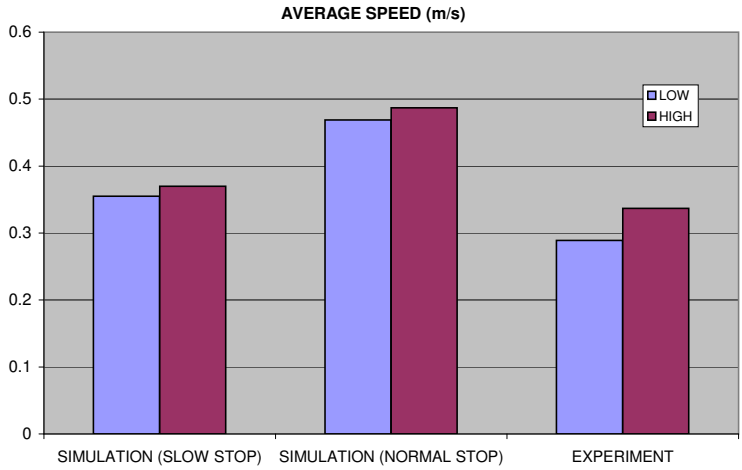


Figure 6.24: Average speeds achieved under different dynamic settings: "simulation (slow stop)" (the third and fourth simulation tests, Chapter 6.4.2), "simulation (normal stop)" (the first and second simulation tests, Chapter 6.4.2), "experiment" (the first and second experimental tests, Chapter 6.4.3).

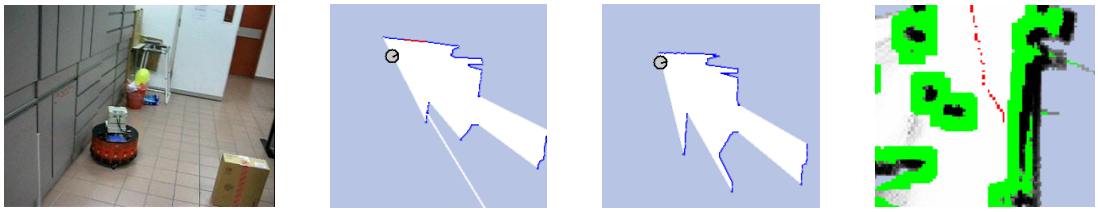
As shown in Figs. 6.19(d) and 6.19(e), the robot passed the door, a narrow passage, between the 4th and 5th searches (more accurately, during the period of approaching the 1st waypoint obtained by the 4th search). The translation velocity profile indicates that the robot accelerated itself when passing the door.

Fig. 6.27 shows the result of another simulation test taken under the "slow stop" setting, where the robot managed to navigate through a series of narrow surroundings and reach the goal.

Since the incremental search algorithm heavily relies on the accuracy of grid maps, the robot may be unable to accurately reach the waypoints or the goal due to localization errors. Fortunately, the robot is able to efficiently avoid collision with obstacles reactively based on sensory data with the proposed approach. Fig. 6.25(h) shows that the cabinets are not properly plotted on the map of C-space, as some misalignment happened, but the robot was able to approach the waypoint without touching the obstacles. Nevertheless, it will be beneficial to implement a localization method other than scan matching used by this research in order to correct localization error before it evolutes to be large.

Though moving obstacles have not been explicitly considered, the proposed approach is able to deal with mobile robot path planning in non-stationary environments consisting of moving obstacles. The allowed travel distance $s_{\text{stop}}(O)$ is reactively computed based on the surroundings and the robot's deceleration capability. Thus any

(a) 2.8 s before stop. (b) 2.1 s before stop. (c) 1.5 s before stop. (d) 0.4 s before stop.



(e) Robot was stopped. (f) 1.7 s before. (g) 0.5 s before. (h) Trajectories.

Figure 6.25: Snapshots of the first experiment before the third search. Diagram (a)-(e): snapshots of the robot in the experiment. Diagram (f)-(g): snapshots of current laser scan and robot's motion direction. Diagram (h): robot trajectories when the robot was stopped.

change of obstacles is able to be converted to proper velocity limit for the robot to perform necessary collision avoidance. The control period and interval of laser scan should be short enough for the robot to react in a timely way against the moving speed of the obstacles. In comparison, there exist motion planning approaches which explicitly consider moving obstacles in a dynamic environment [5, 114–116]. They assume that the trajectories or velocities of the moving obstacles are known *a priori*, or measurable. Nevertheless, if the velocities of the moving obstacles are measurable, the proposed robot system is able to detect potential collision more accurately via intersection test between the predicted extended trajectories and the sensed obstacles expanded by the trajectories (in the same period) of the moving obstacles.

6.5.4 Comparison with Other Approaches

The results of optimized dynamic motion planning compares favorably with those created by the reactive point-to-point target tracing algorithm presented in Chapter 6.4.1, which reactively generates a point-to-point motion command based on the relative position of the target waypoint. Optimized dynamic motion planning produces relatively smooth velocity profiles and relatively high speed. More importantly, it is robust in obstacle avoidance. In contrast, instead of dealing with collision checking directly, point-to-point target tracing attempts to avoid collision by setting a speed

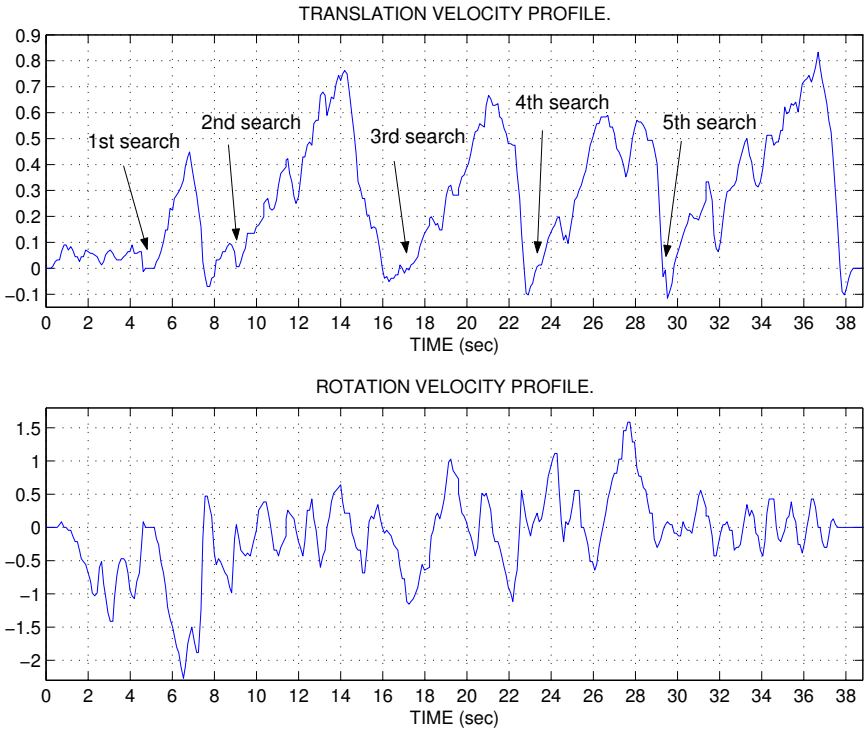
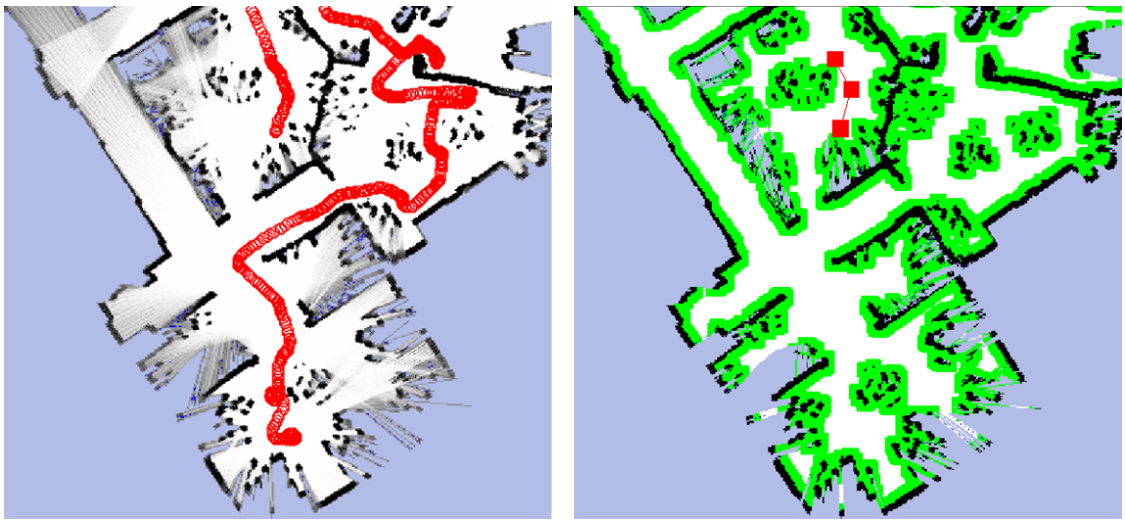


Figure 6.26: Profiles of translation and rotation velocities of the first experiment.

limit as small as 0.3 m/s and by setting a bigger safety margin coefficient c_{enlarge} (1.5 in simulations and 2.0 in experiments). With such measures alone, the robot may still be endangered, as have been found in both simulations and experiments.

The Nearness Diagram algorithm [83] navigates a robot reactively based on situations of the surroundings and the goal's relative position to simplify the difficulty of navigation in troublesome scenarios. The work in [84] further considers shape and kinematics together in an exact manner in its obstacle avoidance process. However, being directional approaches, they may be inadequate to take the robot dynamics into account, which may result in slow or jerky movements.

Dynamic Window approaches and Nearness Diagram assume circular robot motion in a control period and in the period for the robot to stop. Our collision avoidance methodology has taken into account the existence of accelerations in varying the current translation and rotation velocities to the commanded ones. In this way, the predicted extended trajectories could be more accurate, and so is the value of the allowed travel distance $s_{\text{stop}}(O)$ thus computed. Via computing the limit of translation velocity with Eq. (6.19), the obstacle constraints are transformed to suitable velocity limits for the robot to perform relatively high-speed navigation while avoiding collision



(a) Robot's final trajectories and global map. (b) Map for last search.

Figure 6.27: Robot's final trajectories, and search map in a simulation test (robot navigated from bottom to top).

when needed. In comparison, Dynamic Window approaches consider the stop distance based on the chosen velocity command and the assumption of circular robot motion as in Eq. (6.20).

Velocity space approaches (Dynamic Window approaches [86–88] or Curvature-velocity method [90]) typically search in the velocity space for a velocity pair minimizing a single objective function. The optimized motion command found in this way may not be suitable for a real scenario. Our method attempts to use various strategies and objective functions to cater to different situations that the robot is currently in (e.g. the surroundings, and whether the robot is leaving or approaching a waypoint), rather than trying to design a universally applicable single objective function. Each of the weights mentioned in Chapter 6.3.2 was roughly assigned a value (same for all the simulations and experiments) to indicate its significance relative to others. It is found out that such a parameter set, without being fine-tuned, suffices for our program to run successfully in different scenarios and for both simulations and experiments.

In the potential-field-based controllers proposed by Pathak and Agrawal [117], the effects of dynamic constraints on the torque inputs to the robot motors are considered directly using a dynamic model instead of the kinematic model. This might result in a smoother control and a smoother robot trajectory. However, the obstacle constraints have yet to be transformed to suitable torques for the robot to accelerate/decelerate at the right time and in the right magnitude in order for high-speed navigation and

effective collision avoidance. In addition, differential drive mobile robots typically accept translation and velocity control (only), and thus the torque-based control may have to be converted back to velocity control via integration, as in the experiments of [117].

There are a number of learning based methods for mobile robot motion planning, for example, a method to detect dynamic local minima through learning [118], and a probabilistic learning approach [119] to compute the feasible paths for the robots. Some neural-network models [120, 121] were proposed to generate collision-free robot trajectories through learning. However, robot path planning is not efficient and computationally expensive, especially in its initial learning phase. Though requiring a certain amount of time in searching global optimal paths, the modified A* algorithm only need to perform search periodically and inherits the robustness of the A* algorithm in finding a path. Furthermore, the proposed collision avoidance technology enables the robot to move at a relatively high speed based on local sensory information, which makes it robust to change of the environment.

6.6 Summary

This chapter has presented a hierarchical approach for incrementally planning optimal paths and subsequently tracing them in unknown environments. The high-level planner based on the modified A* algorithm is able to handle maps containing unknown information and can robustly plan optimal (possible) paths incrementally. Situation-dependent object functions and strategies, instead of a single objective function, are employed to search for an optimized, waypoint-directed motion in a reduced one-dimensional velocity space. Accelerations in varying translation and rotation velocities are taken into account, and obstacle constraints are transformed to suitable velocity limits for the robot to achieve collision-free, relatively high-speed navigation. Convergence to each target is achieved by ensuring connectivity and a decrease in the robot's distance/orientation angle to the target waypoint when possible. Extensive simulation and experimental results verified the efficacy and robustness of the proposed algorithm in incrementally obtaining a series of optimal paths, and successfully tracing them at a relatively high speed without collision with obstacles. Finally, thorough discussions of the test results and comparisons with other approaches are provided.

Chapter 7

Conclusions and Recommendations

The work presented in this thesis focused on the development of a framework for path planning and motion planning for mobile robots with limited information about the environment and subject to various robot constraints. In this chapter, the results of the research work described in the previous chapters are summarized and the major contributions of this work are reviewed. Suggestions for future work are also presented.

7.1 Summary and Contributions

The research has covered modeling of nonholonomic mobile robots, sensor-based path planning and motion planning, online map building, and simultaneous mapping building and motion planning. It has examined and taken robot constraints into account in motion planning while ensuring the robot's convergence to the goal. In this thesis, an attempt has been made to develop necessary techniques and demonstrate theoretically and empirically that the proposed solutions satisfy the constraints imposed by the overall problem. The principal theoretical and practical results include:

- **Boundary Following Using Instant Goals and Globally Convergent Path Planner (Chapter 3):** A practical approach has been proposed for a holonomic mobile robot to achieve the task of boundary following by continuously locating a series of Instant Goals, instead of assuming that a robot is always able to properly follow an obstacle in an unknown, obstacle-cluttered environment. One significance of this approach is that it may be used as a practical navigation function or behavior that is required by the Bug algorithms and

a number of behavior-based navigation approaches. A potential field method is used to perform reactive collision avoidance, which can be incorporated into boundary following when needed.

Based on the boundary following function and a Bug-like strategy, we propose a realistic sensor-based path planner with global convergence property. The robot navigates in an unstructured, complex environment, and makes decisions based on discrete, and noisy range data. By utilizing all available local information, the path planner decides an optimal local direction to follow an obstacle. Compared to the Bug algorithms, the proposed path planner is more practical as it does not require ideal assumptions such as perfect sensing ability or boundary following capability.

- **PPC Curve for Smooth, Constrained Path Generation for Nonholonomic Robots and Sensor-based Hybrid Path Planning (Chapter 4):**

  The PPC based geometric method generates a robot path that is smooth (continuous) and upper-bounded in curvature, and with a velocity profile satisfying the robot dynamics, which is desirable for path planning for differential drive or car-like robots. The robot constraints and the curvature requirement are examinable for the velocity profile associated with the designed path. Two arbitrary robot configurations, rather than two straight lines, are able to be connected. For real-time, collision-free path planning, a simple yet efficient method utilizing the particular properties of the curve is proposed for collision test of the complex curve.

  The Instant Goal approach was further adapted and improved for differential drive mobile robots by planning smooth paths and considering the robot dynamics. Each time a PPC based curve is constructed to test if it can smoothly connect the current robot position with a candidate Instant Goal, such that the generated path to it is smooth and satisfies the dynamic constraints. The hybrid approach plans a series of motions before actually executing them, while a motion is generated by a fuzzy controller for wall following to overcome the limitation that sometimes the proposed approach might not be able to locate an Instant Goal.

- **A Practical Methodology of Online Map Building for Autonomous Mobile Robots (Chapter 5):** Bayesian theorem is used to update recent

measurements to an occupancy grid map for its efficiency. The application of incremental ML scan matching helps solve the problem of simultaneous localization and map building (SLAM) in a simple way. Sensor fusion of laser and sonar data with a selective method improves map performance in obstacle detection and mapping accuracy. Collision avoidance is also made more robust to obstacles by using both laser and sonar data. Finally, an off-line method is implemented to convert the constructed grid map into a topological one, which may be better suited for large maps and outdoor environment applications.

- **A Hierarchical Framework for Incremental Path Planning and Optimized Dynamic Motion Planning in Unknown Environments (Chapter 6):** This research has established a hierarchical framework, for producing optimal paths robustly with a periodically updated map via deliberative path planning, and for tracing subgoals at a relatively high speed free of collision with obstacles. The following lists the main contributions of the framework:

  i) *A high-level planner based on a modified A* algorithm:* the planner is able to handle a map containing unknown information, and is robust in finding optimal (possible) paths. Furthermore, addition of "obstacle cost" ensures the waypoints planned located not too close to obstacles, which helps subsequent motion planning and collision avoidance.

  ii) *Accelerations in a control period are considered for the first time for dynamic motion planning:* this enables the system to accurately predict the robot pose and trajectory resulting from a velocity command. The such computed allowed travel distance $s_{\text{stop}}(O)$ allows obstacle constraints to be transformed to suitable velocity limits – the robot is thus able to perform high-speed navigation while avoiding collision when needed.

  iii) *Multi-situations are considered in searching for an optimized velocity pair for the first time:* various strategies and objective functions are defined to cater to different navigation situations that the robot is in. In this way, rather than trying to design a universally applicable single objective function, the system is able to handle different situations more easily and robustly. There is no need to fine-tune special parameters/weights to make the optimized motion planning approach work.

iv) *Convergence to waypoints is ensured in local motion planning and thus global convergence to the goal:* in the process of generating a motion command, alignment to the target and convergence to it are considered with high priority, such that the robot is able to reach each subgoal as expected. This is normally not taken into account by velocity space approaches.

7.2 Suggestions for Future Work

After a review of this research work, this section presents the directions that are recommended for extending the results developed in this thesis:

- One limitation of the Instant Goal approach is that it employs a potential field based method for reactively avoiding collision with obstacles. The motions generated may not be smooth, and may not be always feasible for a mobile robot to achieve continuous motions. This problem has been partially addressed in the improved Instant Goal approach by applying a PPC method for smooth, feasible connection. A possible alternative is to apply the collision avoidance algorithm of the hierarchical planning framework to achieve relatively high-speed and smooth navigation.

- When applied to path planning for car-like robots in a sensor-based scenario, the PPC based method magnifies its limitation that it may have a small probability in finding a suitable solution based on limited information. This is because the requirement of a minimum turning radius greatly limits the possible directions of motions of a car-like robot, while the combined curve consisting of a PPC curve or a half PPC curve covers only part of the many possible feasible paths. If partial knowledge of the environment is available, a graph-search method may be used to search for a solution made up by a combined curve in the searched space. This may help enhance the possibility of finding a solution which could be more optimized as well. However, the proposed collision test method might not be applicable since in this case it must be performed in a discrete space.

  In addition, it is assumed that a car-like robot moves at a relatively low speed, which could be acceptable in a laboratory environment. However, for a high-speed vehicle like a car in an outdoor environment, more investigations should be made to ensure that the processes for motion planning and collision test satisfy

the critical requirement of time. A possible solution is to develop an intelligent steering method capable of learning and adapting to the new environmental knowledge, like the behavior of a human being driving a car, such that an autonomous car is able to achieve proper maneuvers in real-time in a dynamic and changing environment.

- Localization errors could manifest themselves when the scale of the environment becomes large, or a cycle of navigation is made. Due to the nature of a graph search algorithm, the modified A* algorithm inevitably relies heavily on the accuracy of the map that it used. Therefore, localization errors may prevent the robot from accurately reaching the goal or a waypoint. An improvement might be made to the current incremental Maximum Likelihood algorithm, such that posterior poses can be corrected once a loop is determined to be closed. This might mean halting the robot exploration so that the global map can be realigned. Possible methods like Expectation Maximization Algorithms can be considered. An alternative is to apply feature (such as corners, straight lines) detection, or even an approach combined by both topological and geometric mapping, to correct these obvious localization errors.

- 3D sensing (sonar, or 3D laser rangefinder) can be implemented in order to improve the robustness of the proposed collision avoidance methodology, especially when the robot is tested in a highly dynamic environment involving moving obstacles or consisting of more unstructured obstacles.

- Our work could be extended to the applications of multiple mobile robots. It is well known that coordination or collaboration among multiple mobile robots could enhance the robustness of a system to a failure of parts of robots and improve the efficiency in accomplishing an assignment through proper task allocation. Yet, the robot constraints, especially the robot dynamics, have seldom been considered in the prevailing research activities on multi-robot path planning or exploration, where typically each robot is assumed to be able to move exactly as commanded. Therefore, it would be interesting to extend our work, especially the hierarchical framework of incremental path planning and optimized dynamic motion planning to multi-robot applications. This extension however might arouse additional problems due to the gap between the nature of multi-robot applications and that of single-robot applications.

Bibliography

[1] J. C. Latombe, *Robot Motion Planning*. London: Kluwer Academic Publishers, 1991.

[2] V. Lumelsky and A. A. Stepanov, "Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape," *Algorithmica*, no. 2, pp. 403–430, 1987.

[3] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 19, no. 5, pp. 1179–1187, 1989.

[4] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1398–1404, Apr 1991.

[5] S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," *Autonomous Robots*, vol. 13, no. 3, pp. 207–222, 2002.

[6] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[7] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278 –288, 1991.

[8] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of of Robotics and Automation*, vol. 2, no. 1, pp. 14–23, 1986.

[9] R. C. Arkin, "Motor-schema-based mobile robot navigation," *The International Journal of Robotics Research*, vol. 8, no. 4, pp. 92–112, 1989.

[10] R. C. Arkin, "Behavior-based robot navigation for extended domains," *Adaptive Behavior*, vol. 1, no. 2, pp. 201–225, 1992.

[11] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, 1992.

[12] J. O. Kim and P. K. Khosla, "Real-time obstacle avoidance using harmonic potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 3, pp. 338–349, 1992.

[13] J. Barraquand and J. C. Latombe, "Robot motion planning: A distributed representation approach," *The International Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649, 1991.

[14] J. Barraquand, L. Kavraki, J. C. Latombe, T. Li, R. Motwani, and P. Raghavan, "A random sampling scheme for path planning," *The International Journal of Robotics Research*, vol. 16, no. 6, pp. 759–774, 1997.

[15] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[16] V. Lumelsky and T. Skewis, "Incorporating range sensing in the robot navigation function," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 5, pp. 1058–1069, 1990.

[17] I. Kamon and E. Rivlin, "Sensory-based motion planning with global proofs," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 6, pp. 814–822, 1997.

[18] I. Kamon, E. Rimon, and E. Rivlin, "Tangentbug: A range-sensor-based navigation algorithm," *The International Journal of Robotics Research*, vol. 17, no. 9, pp. 934–953, 1998.

[19] I. Kamon, E. Rimon, and E. Rivlin, "Range-sensor based navigation in three dimensions," *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 1, pp. 163–169, 1999.

[20] S. Rajko and S. M. LaValle, "A pursuit-evasion bug algorithm," *Proceedings of 2001 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1954–1960, 2001.

[21] B. H. Krogh and D. Feng, "Dynamic generation of subgoals for autonomous mobile robots using local feedback information," *IEEE Transation on Automatic Control*, vol. 34, no. 5, pp. 483–493, 1989.

[22] H. Noborio, I. Yamamoto, and T. Komaki, "Sensor-based path-planning algorithms for a nonholonomic mobile robot," *Proceeding of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 917–924, 2000.

[23] T. Yata, L. Kleeman, and S. Yuta, "Wall following using angle information measured by a single ultrasonic transducer," *Proceedings of 1998 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1590–1596, May 1998.

[24] A. Bemporad, M. D. Marco, and A. Tesi, "Sonar-based wall-following control of mobile robots," *ASME J. Dynamic Systems, Measurement and Control*, vol. 122, pp. 226–230, 2000.

[25] J. Taheri and N. Sadati, "A fully modular online controller for robot navigation in static and dynamic environments," *Proceedings of 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, vol. 1, pp. 163–168, 2003.

[26] M. J. Matarić, *A distributed model for mobile robot environment-learning and navigation.* Master's thesis, MIT, Cambridge, MA, Jan 1990.

[27] H. Choset, *Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph.* PhD thesis, California Institute of Technology, 1996.

[28] J.-S. Gutmann, T. Weigel, and B. Nebel, "A fast, accurate, and robust method for selflocalization in polygonal environments using laser-range-finders," *Advanced Robotics Journal*, vol. 14, no. 8, pp. 651–668, 2001.

[29] P. Newman, J. Leonard, J. Neira, and J. Tardós, "xplore and return: Experimental validation of real time concurrent mapping and localization," *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002.

[30] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Journal of Robotics and Automation*, vol. 3, no. 3, pp. 249 –265, 1987.

[31] A. Elfes, *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Carnegie Mellon University, 1989.

[32] H. P. Moravec, "Sensor fusion in certainty grids for mobile robots," *AI Magazine*, vol. 9, no. 2, pp. 61–74, 1988.

[33] H. P. Moravec and A. Elfes, "High resolution maps from wide angle sonar," *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 2, pp. 116–121, Mar 1985.

[34] F. Gambino, G. Ulivi, and M.Vendittelli, "The transferable belief model in ultrasonic map building," *Proceedings of 6th IEEE International Conference on Fuzzy Systems*, vol. 2, pp. 601–608, Jul 1997.

[35] D. Pagac, E. Nebot, and H. Durrant-Whyte, "An evidential approach to map-building for autonomous vehicles," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 4, pp. 623–629, 1998.

[36] G. Oriolo, G. Ulivi, and M. Vendittelli, "Fuzzy maps: A new tool for mobile robot perception and planning," *Journal of Robotic Systems*, vol. 14, no. 3, pp. 179–197, 1997.

[37] G. Oriolo, G. Ulivi, and M. Vendittelli, "Real-time map building and navigation for autonomous robots in unknown environments," *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, vol. 28, no. 3, pp. 318–333, 1998.

[38] M. Ribo and A. Pinz, "A comparison of three uncertainty calculi for building sonar-based occupancy grids," *Proceedings of 7th International Symposium on Intelligent Robotic Systems*, pp. 235–243, Jul 1999.

[39] S. Thrun, W. Burgard, and D. Fox, "A probabilistic approach to concurrent mapping and localization for mobile robots," *Autonomous Robots*, vol. 5, no. 3, pp. 253–271, 1998.

[40] J. Castellanos, J. Montiel, J. Neira, and J. Tardós, "The spmap: A probabilistic framework for simultaneous localization and map buildingn," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 5, pp. 948–953, 1999.

[41] G. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localisation and map building (slam) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–241, 2001.

[42] O. Karch and H. Noltemeier, "Intelligent robots and systems," *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, pp. 850–856, Sep 1997.

[43] J. A. Castellanos and J. D. Tardós, *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach*. Norwell, Massachusetts: Kluwer Academics Publishers, 1999.

[44] J. Guivant and E. Nebot, "Optimization of the simultaneous localization and map building algorithm for real time implementation," *IEEE Transactions of Robotic and Automation*, vol. 17, no. 3, pp. 242–257, 2001.

[45] H. Durrant-Whyte, S. Majumder, S. Thrun, M. de Battista, and S. Sheding, "Bayesian algorithm for simultaneous localization and map building," *Proceedings of the 10th International Symposium of Robotics Research (ISRR01)*, pp. 249 –265, Jun 2001.

[46] S. Thrun, "Robotic mapping: A survey," *In G. Lakemeyer and B. Nebel, editors, Exploring Artificial Intelligence in the New Millenium*, 2002.

[47] G. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*. New York: Wiley Series in Probability and Statistics, 1997.

[48] F. Lu and E. Milios, "Optimal global pose estimation for consistent sensor data registration," *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1, no. 1, pp. 93 – 100, May 1995.

[49] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, p. 333C349, 1997.

[50] F. Lu and E. Milios, "Robot pose estimation in unknown environments by matching 2d range scansg," *Journal of Intelligent and Robotic Systems*, vol. 18, pp. 249–275, 1997.

[51] D. Hähnel, W. Burgard, D. Fox, and S. Thrun, "An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," *IEEE/RSJ International Conference on Intelligent Robots and System, 2003*, vol. 1, pp. 206 – 211, Oct. 2003.

[52] D. Hähnel, D. Schulz, and W. Burgard, "Mobile robot mapping in populated environments," *Advanced Robotics*, vol. 17, no. 7, pp. 579–597, 2003.

[53] N. J. Nilsson, *Principles of Artificial Intelligence*. Tioga Publishing Company, Jan 1980.

[54] D. Kortenkamp, R. Bonasso, and R. Murphy, *Artificial Intelligence and Mobile Robots*. AAAI Press/The MIT Press, 1998.

[55] L. Dorst and K. I. Trovato, "Optimal path planning by cost wave propagation in metric configuration space," *Mobile Robotics III, SPIE proceedings*, vol. 1007, pp. 186–197, 1989.

[56] S. S. Keerthi, C. Ong, E. Huang, and E. Gilbert, "Equidistance diagram – a new roadmap method for path planning," *Proceedings of 1999 IEEE Conference on Robotics and Automation*, pp. 682–687, 1999.

[57] J. Barraquand and J. C. Latombe, "Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles," *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2328–2335, 1991.

[58] Y. Hwang, P. Xavier, P. Chen, and P. Watterberg, *Motion planning with SANDROS and the configuration space toolkit, In K.K. Gupta and A.P. del Pobil, editors, Practical Motion Planning in Robotics*. John Wiley & Sons, 1998.

[59] J. Barraquand and J. C. Latombe, "On nonholonomic mobile robots and optimal maneuvering," *Revue d'Intelligence Artificielle*, vol. 3, no. 2, pp. 77–103, 1989.

[60] H. Choset and J. Burdick, "Sensor based planning, part ii: Incremental construction of the generalized voronoi graph," *Proceedings of the IEEE International Conference on Robotics and Automation*, 1995.

[61] H. Choset and J. Burdick, "Sensor-based exploration: The hierarchical generalized voronoi graph," *The International Journal of Robotics Research*, vol. 19, no. 2, pp. 96–125, 2000.

[62] A. Stentz, "Optimal and efficient path planning for partially-known environments," *Proceedings of the IEEE Conference on Robotics and Automation*, pp. 3310–3317, 1994.

[63] A. Stentz, "The focussed d* algorithm for real-time replanning," *Proceedings of International Joint Conference on Artificial Intelligence*, 1995.

[64] A. Yahja, S. Singh, and A. Stentz, "An efficient on-line path planner for outdoor mobile robots operating in vast environments," *Robotics and Autonomous Systems*, vol. 32, no. 2&3, pp. 129–143, 2000.

[65] F. Pin and S. Killough, "A new family of omnidirectional and holonomic wheeled platforms," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 4, pp. 480–489, 1994.

[66] K. Watanabe, "Control of an omnidirectional mobile robot," *2nd Int. Conf. Knowledge-Based Intell. Electro. Sys.*, pp. 51–60, 1998.

[67] T. Yamada, K. Watanabe, and K. Kiguchi, "Dynamic model and control for a holonomic omnidirectional mobile robot," *Autonomous Robots*, vol. 11, no. 2, pp. 173–189, 2001.

[68] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, pp. 497–517, 1957.

[69] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[70] P. Jacobs and J. Canny, "Planning smooth paths for mobile robots," *Proceedings of 1989 IEEE International Conference on Robotics and Automation*, vol. 1, pp. 2–7, May 1989.

[71] J.-P. Laumond, P. Jacobs, M. Taix, and R. Murray, "A motion planner for nonholonomic mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 5, pp. 577 – 593, 1994.

[72] J.-P. Laumond, S. Sekhavat, and F. Lamiraux, *Guidelines in nonholonomic motion planning, in J.-P. Laumond, editor, Robot motion planning and control*. Berlin, DE: Springer-Verlag, 1998.

[73] Y. Kanayama and N. Miyake, "Trajectory generation for mobile robots," *Robotics Research*, vol. 3, pp. 333–340, 1986.

[74] S. Fleury, P. Soueres, J.-P. Laumond, and R. Chatila, "Primitives for smoothing mobile robot trajectories," *Proceedings of 1993 IEEE International Conference on Robotics and Automation*, vol. 1, pp. 832–839, May 1993.

[75] A. Scheuer and T. Fraichard, "Continuous-curvature path planning for car-like vehicles," *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, pp. 997 – 1003, Sep. 1997.

[76] T. Fraichard and A. Scheuer, "From reeds and shepp's to continuous-curvature paths," *IEEE Transactions on Robotics*, vol. 20, no. 6, pp. 1025–1035, 2004.

[77] Y. Kanayama and B. I. Hartman, "Smooth local path planning for autonomous vehicles," *The International Journal of Robotics Research*, vol. 16, no. 3, pp. 263–283, 1997.

[78] W. Nelson, "Continuous-curvature paths for autonomous vehicles," *Proceedings of 1989 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 1260–1264, May 1989.

[79] O. Pinchard, A. Liegeois, and F. Pougnet, "Generalized polar polynomials for vehicle path generation with dynamic constraints," *Proceedings of 1996 IEEE International Conference on Robotics and Automation*, vol. 1, pp. 915–920, April 1996.

[80] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha, "Fast distance queries with rectangular swept sphere volumes," *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 4, no. 6, pp. 3719–3726, April 2000.

[81] S. A. Cameron, "Collision detection by four-dimensional intersection testing," *IEEE Transactions on Robotics and Automation*, vol. 6, pp. 291–302, 1990.

[82] M. C. Lin and J. F. Canny, "A fast algorithm for incremental distance calculation," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1008–1014, 1991.

[83] J. Minguez and L. Montano, "Nearness diagram (nd) navigation: Collision avoidance in troublesome scenarios," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 1, pp. 45–59, 2004.

[84] J. Minguez and L. Montano, "Abstracting vehicle shape and kinematic constraints from obstacle avoidance methods," *Autonomous Robots*, vol. 20, pp. 43–59, 2006.

[85] F. Lamiraux, D. Bonnafous, and O. Lefebvre, "Reactive path deformation for nonholonomic mobile robots," *IEEE Transactions on Robotics*, vol. 20, no. 6, pp. 967–977, 2004.

[86] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[87] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 1, pp. 341–346, May 1999.

[88] P. Ogren and N. Leonard, "A tractable convergent dynamic window approach to obstacle avoidance," *Proceeding of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, pp. 595–600, 2002.

[89] M. Seder, K. Macek, and I. Petrovic, "An integrated approach to real-time mobile robot control in partially known indoor environments," *In Proceeding of the 31st Annual Conference of the IEEE Industrial Electronics Society*, 2005.

[90] R. Simmons, "The curvature-velocity method for local obstacle avoidance," *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 4, pp. 22–28, April 1996.

[91] N. Y. Ko and R. Simmons, "The lane-curvature method for local obstacle avoidance," *Proceedings of 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, pp. 1615 –1621, Oct. 1998.

[92] Y. Wang and D. M. Lane, "Solving a generalized constrained optimization problem with both logic and and or relationships by a mathematical transformation and its application to robot motion planning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 30, no. 4, pp. 525–536, 2000.

[93] J. K. Rosenblatt and C. E. Thorpe, "Combining multiple goals in a behavior-based architecture," *Proceedings of 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, pp. 136–141, 1995.

[94] M. Fliess, J. Lévine, P. Martin, and P. Rouchon, "Flatness and defect of non-linear systems: introductory theory and examples," *International Journal of Control*, vol. 61, no. 6, pp. 1327–1361, 1995.

[95] W. Nelson, "Continuous steering-function control of robot carts," *IEEE Transactions on Industrial Electronics*, vol. 36, no. 3, pp. 330–337, 1989.

[96] B. d'Andréa Novel, G. Bastin, and G. Campion, "Modelling and control of non-holonomic wheeled mobile robots," *Proceedings of 1991 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1130–1135, April 1991.

[97] G. Campion, G. Bastin, and B. d'Andréa Novel, "Structural properties and classification of kinematic and dynamic models of wheeled mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 1, pp. 47–62, 1996.

[98] S. S. Ge, Y. J. Cui, and C. Zhang, "Instant-goal-driven methods for behavior-based mobile robot navigation," *Proceedings of 2003 IEEE International Symposium on Intelligent Control*, pp. 269–274, Oct 2003.

[99] Y. K. Hwang and N. Ahuja, "Gross motion planning — a survey," *ACM Computing Surveys*, vol. 24, no. 3, pp. 219–291, 1992.

[100] P. Malkin and S. Addanki, "Lognets: A hybrid graph spatial representation for robot navigation," *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pp. 1045–1050, 1990.

[101] M. Piaggio and A. Sgorbissa, "Ai-cart: an algorithm to incrementally calculate artificial potential fields in real-time," *1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pp. 238–243, 1999.

[102] S. L. Tan and J. Gu, "Investigation of trajectory tracking control algorithms for autonomous mobile platforms: Theory and simulation," *Proceedings of the 2005 International Conference on Mechatronics and Automation*, vol. 2, pp. 934–939, 2005.

[103] M. Mucientes and J. Casillas, "Obtaining a fuzzy controller with high interpretability in mobile robots navigation," *Proceedings of IEEE International Conference on Fuzzy Systems*, vol. 3, pp. 1637–1642, 2004.

[104] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998.

[105] J. Foley, A. Dam, S. Feiner, and J. Hughes, *Computer Graphics, Principles And Practice, second ed.* Addison Wesley, 1990.

[106] S. Thrun, "A probabilistic online mapping algorithm for teams of mobile robots," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 335–363, 2001.

[107] J. H. Lim and D. W. Cho, "Specular reflection probability in the certainty grid representation," *Transaction of the ASME Jour. of Dynamic System, Measurements and Control*, vol. 116, pp. 512–520, 1994.

[108] iRobot Inc., *Mobility Robot Integration Software User's Guide, Part Number: 2841; Rev. 5.* iRobot, Inc., USA., 2002.

[109] R. Szabo, "Topological navigation of simulated robots using occupancy grid," *International Journal of Advanced Robotic Systems*, vol. 1, no. 3, pp. 201–206, 2004.

[110] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Communications of ACM*, vol. 27, no. 3, 1984.

[111] R. Biswas, B. Limketkai, S. Sanner, and S. Thrun, "Towards object mapping in dynamic environments with mobile robots," *Submitted for publication*, 2002.

[112] Y. Hao and S. K. Agrawal, "Formation planning and control of ugvs with trailers," *Autonomous Robots*, vol. 19, no. 3, pp. 257–270, 2005.

[113] X. C. Lai, S. S. Ge, P. T. Ong, and A. A. Mamun, "Incremental path planning using partial map information for mobile robots," *Proceedings of the 9th International Conference on Control, Automation, Robotics and Vision, ICARCV 2006*, pp. 133–138, 5-8th December 2006.

[114] K. Fujimura and H. Samet, "A hierarchical strategy for path planning among moving obstacles," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 1, pp. 61–69, 1989.

[115] N. Y. Ko and B. H. Lee, "Avoidability measure in moving obstacle avoidance problem and its use for robot motion planning," *Proc. IEEE/RSJ Int. Conf. Intell. Robots and Sys.*, vol. 3, pp. 1296 –1303, 1996.

[116] R. A. Conn and M. Kam, "Robot motion planning on n-dimensional star worlds among moving obstacles," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 320–325, 1998.

[117] K. Pathak and S. K. Agrawal, "An integrated path-planning and control approach for nonholonomic unicycles using switched local potentials," *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1201–1208, 2005.

[118] L. M. Gambardella and C. Versino, "Robot motion planning integrating planning strategies and learning methods," *Proceedings of 2nd IEEE International Conference on AI Planning Systems*, June 1994.

[119] P. Svestka and M. H. Overmars, "Motion planning for carlike robots using a probabilistic approach," *International Journal of Robotics Research*, vol. 16, no. 2, pp. 119–145, 1997.

[120] E. Zalama, P. Gaudiano, and J. L. Coronado, "A real-time, unsupervised neural network for the low-level control of a mobile robot in a nonstationary environmen," *Neural Network*, vol. 8, no. 1, p. 103C123, 1995.

[121] A. Willms and S. Yang, "An efficient dynamic system for real-time robot-path planning," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 36, no. 4, pp. 755– 766, 2006.

[122] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: the carnegie mellon navigation (carmen) toolkit," *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, pp. 2436–2441, Oct. 2003.

[123] R. Simmons and G. Whelan, "Visualization tools for validating software of autonomous spacecraft," *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, July 1997.

[124] R. Simmons and D. James, *Inter-process communication: a reference manul*. Carnegie Mellon University, 2005.

Appendix A

Frame Transformation

In implementations, it is often necessary to convert robot or goal coordinates between global (Fig. A.1(a)), localized (Fig. A.1(b)) or grid coordinate frames. For easier formulation, a 3rd row is added to 2D coordinates, such that the position is expressed by $P = (x, y, 1)$. In Fig. A.1(c), it is known that the coordinates of a point (say C) in frame A, ${}^{A}P_C$, can be obtained if we know its coordinates in frame B, ${}^{B}P_C$, and the translation (x_{AB}, y_{AB}) and the rotation (theta θ) from frame A to frame B. The transformation is given by:

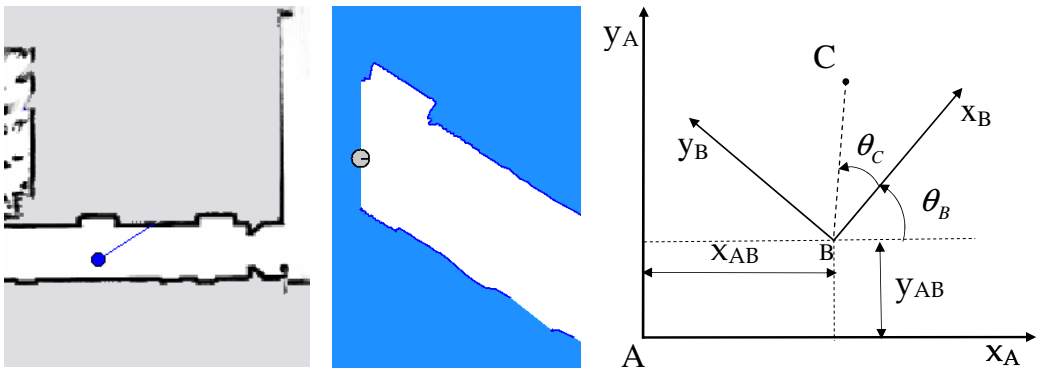
$$\begin{bmatrix} {}^{A}x_C \\ {}^{A}y_C \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & x_{AB} \\ \sin\theta & \cos\theta & y_{AB} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{B}x_C \\ {}^{B}y_C \\ 1 \end{bmatrix}. \tag{A.1}$$

If ${}^{A}_{B}\mathbf{R}$ represents the 2D homogeneous transformation (translation and rotation) to shift frame A to frame B, Eq. (A.1) can be expressed in the following compact form:

$${}^{A}P_C = {}^{A}_{B}\mathbf{R}\ {}^{B}P_C, \tag{A.2}$$

where ${}^{A}_{B}\mathbf{R}$ can be also considered as the origin of frame B and the rotation vector of frame B in frame A.

In our programs to implement the proposed mapping or motion planning algorithms, the types of the robot coordinate systems include the global, simulator and localized ones, among others. Since the initial simulator pose and the initial localized pose w.r.t. the global frame are both known, the transformation matrix from the simulator frame and the localized frame to the global frame can be obtained as ${}^{G}_{S}\mathbf{R}$ and ${}^{G}_{L}\mathbf{R}$, respectively. Therefore, the coordinates of point C can be obtained as



(a) Global frame. (b) Robot frame. (c) Frame Transformation.

Figure A.1: Global and localized frames (robot is at its initial robot pose).

follows:

$$^{G}P_C = {}^{G}_{S}\mathbf{R}\,{}^{S}P_C = {}^{G}_{L}\mathbf{R}\,{}^{L}P_C. \tag{A.3}$$

In view of the above, the transformation from simulator coordinates ${}^{S}P_C$ (which is known) to localized coordinates can be done as follows:

$$^{L}P_C = ({}^{G}_{L}\mathbf{R})^{-1}\,{}^{G}_{S}\mathbf{R}\,{}^{S}P_C. \tag{A.4}$$

If the location to place the first localized pose in an occupancy grid map is known, the transformation matrix from the localized frame to the occupancy grid frame can be obtained as ${}^{O}_{L}\mathbf{R}$. Therefore, the transformations between localized coordinates ${}^{L}P_C$ and grid coordinates ${}^{O}P_C$ can be given by:

$$\begin{cases} {}^{O}P_C = {}^{O}_{L}\mathbf{R}\,{}^{L}P_C \\ LP_C = ({}^{O}_{L}\mathbf{R})^{-1}\,{}^{O}P_C. \end{cases} \tag{A.5}$$

Suppose that a grid map's resolution is γ (in unit m^2/cell) and its size is $o_{\max}$ m^2. If the middle point $({}^{L}x_m, {}^{L}y_m)$ of the initial robot position and the goal is located at the center of the grid map and if there is no rotation during transformation, the transformation from localized coordinates ${}^{L}P_C$ to grid coordinates ${}^{O}P_C$ is:

$$^{O}P_C = \frac{1}{\gamma}\begin{bmatrix} 1 & 0 & \gamma\frac{o\max}{2} - {}^{L}x_m \\ 0 & 1 & \gamma\frac{o\max}{2} - {}^{L}y_m \\ 0 & 0 & 1 \end{bmatrix}\,{}^{L}P_C. \tag{A.6}$$

The inverse transformation of (A.6) can be given by

$$^{L}P_C = \gamma\begin{bmatrix} 1 & 0 & \frac{{}^{L}x_m}{\gamma} - \frac{o\max}{2} \\ 0 & 1 & \frac{{}^{L}y_m}{\gamma} - \frac{o\max}{2} \\ 0 & 0 & 1 \end{bmatrix}\,{}^{O}P_C. \tag{A.7}$$

Appendix B

Robot System for Experiments

B.1 Hardware System

A Magellan Pro mobile robot, manufactured by iRobot Inc., was used to carry out experimental tests for the proposed map building, motion planning and path planning algorithms. The robot is 40.6 cm in diameter and 25.4 cm in height, with a payload of 9.1 kg. Its weight is about 16 kg (not inclusive of additional components such as a laser scanner). The computer system for a Magellan Pro mobile robot is an on-board Pentium-based EBX (Embedded Board eXchange) system, a single-board computer for embedded applications. It offers functionality equivalent to a conventional PC. Our robot is equipped with an Intel Pentium III 962 MHz CPU and 128M Memory. It can communicate with other computers via ethernet interface as well as serial interface. A picture of the robot is shown in Fig. B.1.

There are 16 sonars (with a beam width of 30 degrees), infra-red (IR), and Bumper sensors equally distributed around the robot base. The Magellan Pro robot comes with 2 lead acid batteries which supplies power for 2-3 hours (when no laser scanner is used). Two 24V DC servomotors are used to differentially drive the robot. This two wheel differential design gives it the ability to turn on the spot. It has a maximum translational speed of 1 m/s and rotational speed of 120°/s. The readers may refer to the manuals and user's guide of Magellan Pro robots [108] for more details.

Fig. B.2 is a top view schematic of a Magellan Pro robot, showing the rFLEX (see next section) screen, emergency stop button, joystick port and charger port. An

Figure B.1: Picture of the Magellan Pro robot.

Orinoco Ethernet Converter 10BaseT (the box in Fig. B.1) is connected to the ethernet on the robot's mother board. Inside the box is an Orinoco Classic Gold Card 11b manufactured by Proxim. The robot is thus able to perform wireless communication with other computers via an access point, Cisco Airnet 340 Series Base Station.

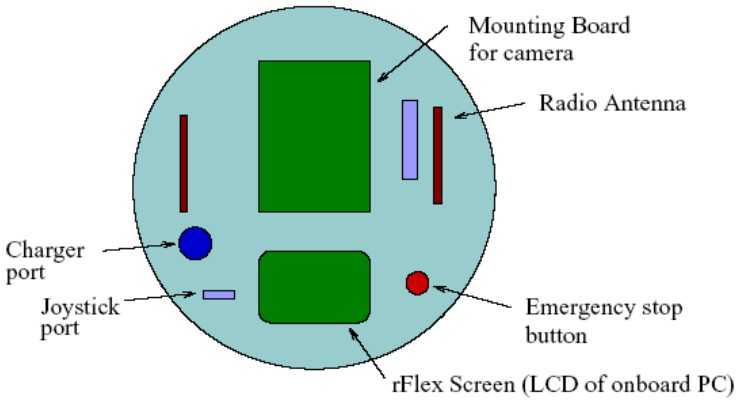


Figure B.2: Top view schematic of a Magellan Pro robot, with the front of the robot facing to the top of the diagram.

On top of the robot, a laser rangefinder, SICK LMS (Laser Measurement System) 291, is installed facing the front, and can scan the surroundings two dimensionally. The device provides relatively accurate and unambiguous sensor readings (see Table B.1). Such a system can be used for various applications, including area monitoring, object measurement and detection and determining positions.

Table B.1: Technical Specifications for SICK LMS 291.

Max Scanning Angle	Resolution / Typical Measurement Accuracy	Angular Resolution	Typical Range with 10% Reflectivity
180°	10mm / ± 60mm	0.25/0.5/1°	30m

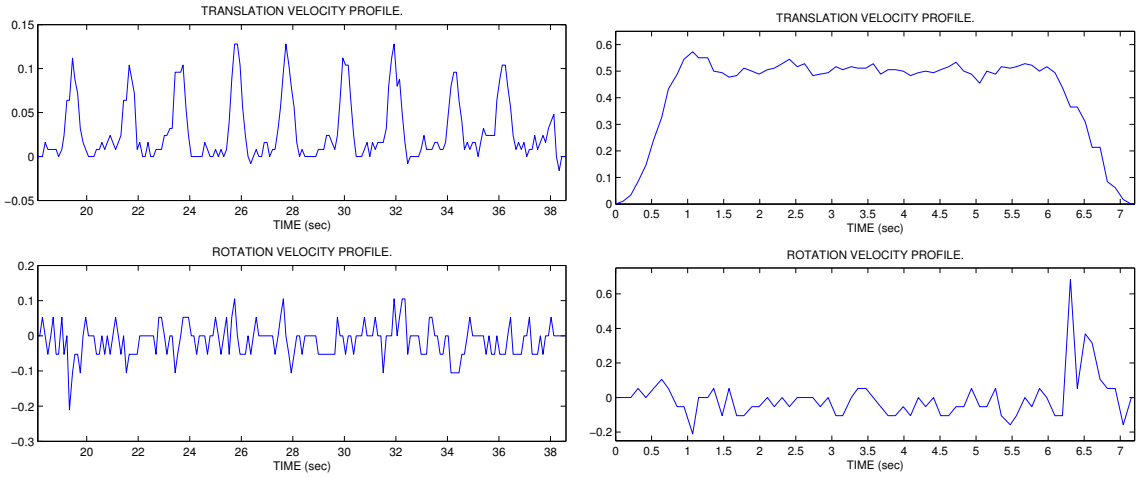
B.2 rFLEX

The user interface of the robot is the rFLEX control system, which is hardware-control software flashed in robot's ROM. rFLEX is used to debug and monitor the hardware system such as sonar sensors, motors. rFLEX has its own monitor screen installed on board.

Pre-filtering is applied before raw sonar data is used. Since the standard Polaroid sonar sensors never return values that are closer than the closest obstacle [108], we can take the closest reading in the recent readings. In our experiments, the latest two readings are taken to find out the correct sonar reading. Since the time interval between two odometry readings (including sonar data) is as small as 0.1 s, the system is still able to obtain pretty recent sonar data, while the erroneous large values can be filtered out.

Magellan Pro robots (and other rFLEX based robot) will stop if they do not receive a command for a certain length of time. A number of experiments were carried out to find out a suitable interval before two motion commands. Fig. B.3(a) shows the resulting velocity profiles when velocity command (0.5, 0) was continuously sent to the robot for 20 s only upon a receipt of laser scan, i.e. at an interval of about every 0.2 s. Fig. B.3(b) shows the result by sending the same velocity commands upon a receipt of base message, i.e. about every 0.1 s. It can be seen that in the former case, the robot first attempted to accelerate to the desired velocity, but it then tended to stop itself since it did not receive a command for a while. It was observed that the robot was unable to track the desired translation velocity. Therefore, the robot should be updated with motion commands every 0.1 s.

Since the units used by the rFLEX for odometry appear to be arbitrary and different on each model of robots, this coefficient is needed to convert to meters: m = (rFLEX units) / (odo_distance_conversion). The robot's conversion factor can be determined by driving the robot for a known distance and observing the output of the rFLEX through a few experimental tests. Fig. B.4(a) shows the resulting velocity



(a) Interval at 0.2 s.

(b) Interval at 0.1 s.

Figure B.3: Velocity profiles of experimental tests on command interval.

profiles when velocity command (0.3, 0) was continuously sent to the robot upon a receipt of base message. Finally, the coefficient's value is determined as follows by comparing the actual traveled distance and the commanded translation velocity times the execution time:

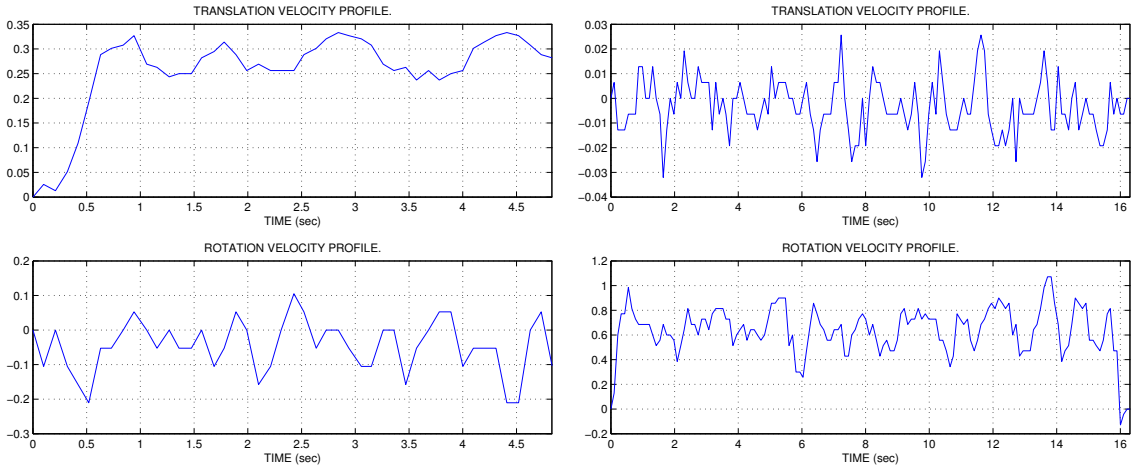
$$odo\_distance\_conversion = 39009.7.$$

Conversion coefficient for rotation odometry can be obtained in a way similar to that of odo_distance_conversion. Fig. B.4(b) shows the resulting velocity profiles when velocity command (0, 45°) was continuously sent to the robot upon a receipt of base message. The coefficient's value is determined as:

$$odo\_distance\_conversion = 5834.0.$$

Another reason for a need of determining the robot's conversion factors is that ground condition also has an effect on the actual velocities that can be achieved by the robot. In addition, Figs. B.3 and B.4 show that the actual translation or rotation velocity were not zero even it was commanded to. This is partially attributed to the disturbances of the outside environment and the effect of the castor wheel's movements on robot motions.

The operating system of a Magellan Pro mobile robot is RedHat 6.2 Linux system. The robot comes installed with MOBILITY^{TM} software for data acquisition and robot control. The MOBILITY is an Object-Oriented software, CORBA-based Robotics Control Architecture. The Mobility Object Manager (MOM) is MOBILITY's



(a) Result of translation velocity commands.

(b) Result of rotation velocity commands.

Figure B.4: Velocity profiles of experimental tests on translation or rotation velocity commands.

Java-based graphical user interface. We can observe, tune, configure and debug Mobility robot control environment, as programs are running. MOM can run on Linux, Windows95, and Windows NT. MOM shows any kind of data such as graphical reading, sonar data, images and text.

Appendix C

Software Package

Instead of using the MOBILITY software that comes with the robot, we choose the Carnegie Mellon Robot Navigation Toolkit (CARMEN) [122] to develop and implement the proposed algorithems presented in Chapters 4, 5 and 6. One reason is that CARMEN is an open-source software package providing useful functions, such as data logging, off-line map building, and simulator. In addition, it also includes its own rFLEX driver to acquire sensory data and to control a Magellan Pro robot.

C.1 IPC for Inter-process Communication

CARMEN uses the inter-process communication platform, the IPC software package developed by Reid Simmons [123]. An IPC-based system consists of an application independent central server and any number of application-specific processes. As stated in IPC manual [124], the basic IPC communication package is essentially a publish/subscribe model, where tasks/processes indicate their interest in receiving messages of a certain type, and when other tasks/processes publish messages, the subscribers all receive a copy of the message. Since message reception is asynchronous, each subscriber provides a callback function (a "handler") that is invoked for each instance of the message type. Tasks/processes can connect to the IPC network, define messages, publish messages, and listen for (and process) instances of messages to which they subscribe.

To facilitate passing messages containing complex data structures, IPC provides utilities for marshalling a C or LISP data structure into a byte stream, suitable for publication as a message, and for unmarshalling a byte stream back into a C or LISP

data structure by the subscribing handlers.

C.2 System Architecture and Modules

Fig. C.1 shows a list of modules of the CARMEN-based software system to run for experimental tests on a robot. Those modules in colored boxes, i.e. "motion planning", "scan matching" and "online mapping", are newly built in this research. The system can be categorized into the following modules:

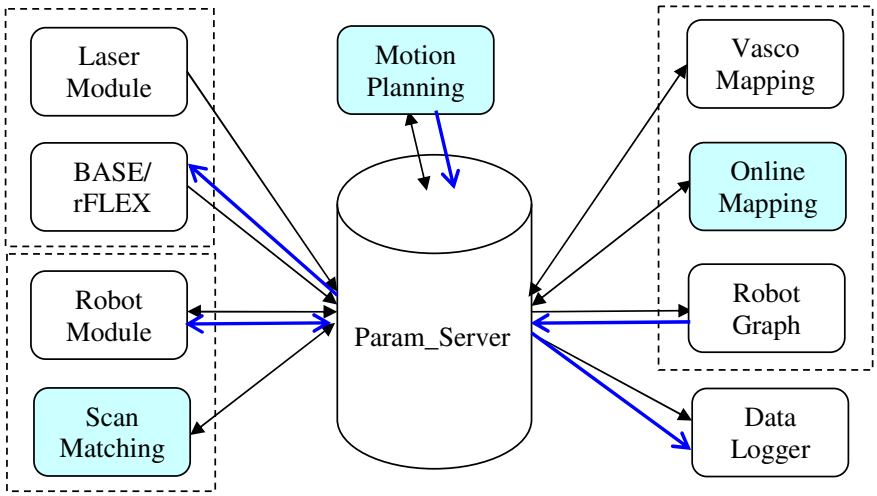


Figure C.1: Main modules of the software architecture used for experimental tests.

1: **Centralized Parameter Server.** *ParamServer* module provides other modules with information (position, sensor data, map, etc.) about the robot. Behind it is the "central" program, which enables communication between these other programs. It keeps track of what is published and delivers it to the subscribers.

2: **Base Services.** This category of programs controls the movement of the robot and accepts inputs from the sensors. These programs should run on the computer attached to the robot hardware.

- *Base* module (rFLEX in our case) provides raw odometry data and may provide sonar data, bumper data and IR data.
- *Laser* module provides laser data obtained from a laser rangefinder.

3: **Graphics Display.** Modules involved in graphics display are recommended to run on other computers instead of the computer attached to the robot hardware, because much time may be needed for a computer to display and (frequently) update graphics.

- *Vasco* module creates a map from sensor and odometry data stored in a log file.
- *Online Mapping* module creates maps online from sensor and odometry data received from ParamServer.
- *Robotgraph* module provides a simple graphic interface for the robot, allowing direct motion control and a display of current sensor information.

4: **Message Transformation.** This category of programs transforms data/message into another format.

- *Robot* module wraps sensor and odometry data into a new format that is convenient for a use in mapping or motion planning.
- *Scan Matching* module does scan matching online to estimate the best robot pose.

5: **Autonomous Navigation** Such programs produce motion instructions based on sensor data and/or map information of the environment to enable autonomous navigation. *Motion Planning* enables both reactive autonomous navigation based on current sensor data, and deliberative path planning with partial map.

6: **Data Logging** *Data Logger* module stores sensor and odometry data, and motion instructions with time stamps into a log file.

The small arrows in Fig. C.1 indicate the flow of data to and from the ParamServer. Taking the advantage that IPC provides flexible, efficient message passing between processes, all modules are able to access the latest data (e.g. sensor readings, grid maps, way points, or goal position).

Sensor data comes from one of two sources: the base module (rFLEX in our case) and the laser module. While it is (obviously) possible to send messages directly to the base module, this is not an exposed interface. Instead, motion commands are first

processed by the "robot" module before reaching the base module. The big arrows in Fig. C.1 indicate the flow of motion instructions to the robot to and from the ParamServer. Data logger provides functions to log all kinds of messages including motion instructions.

Fig. C.2 plots the block diagrams of the CARMEN architecture to run in our experiments. From this diagram, we can clearly see the data flow between different processes. Note that vasco off-line mapping is not included. Data logger module is not necessary but it is beneficial to record sensory data for analysis carried out off-line.

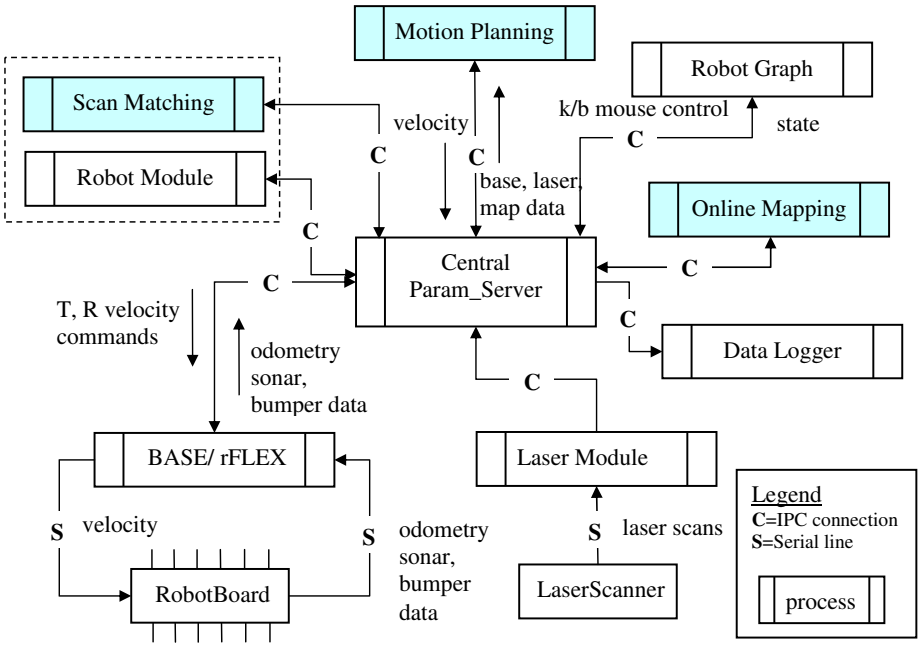


Figure C.2: Block diagrams of the CARMEN architecture for experimental tests involving simultaneous mapping and path planning.

Fig. C.3 shows a list of modules of the CARMEN-based software system to run on a simulated robot for simulation tests.

- *Simulator* module provides simulated data generation from a virtual robot. It requires a pre-built map to represent the environment.
- *Navigation Panel* module is a graphic interface which shows the robot's position and the destination on the pre-built map. It also allows setting of the current robot position and the robot orientation, and selection of the destination.

It is noted that the base and laser modules that are used in an experiment running are replaced by the simulator module. The robot module is not needed any more, since the message/data generated by the simulator module is already in the form of robot messages. In addition, a graphics module "navigation panel" is now provided for settings of the robot pose and destination.

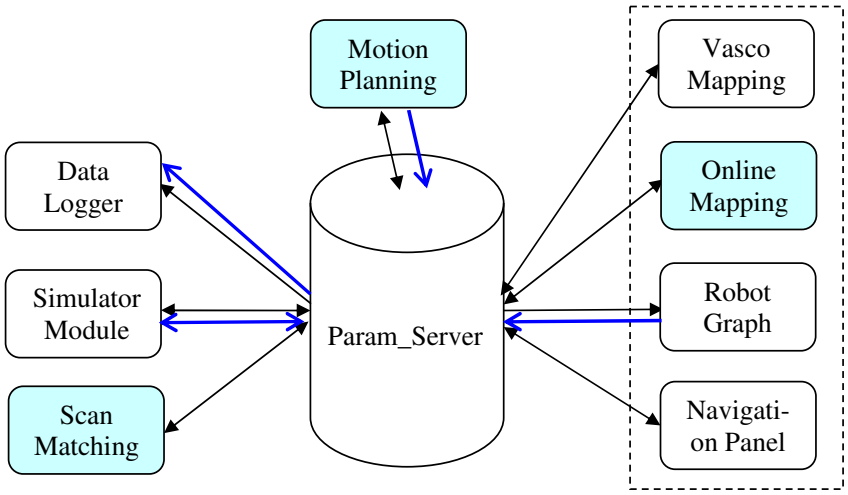


Figure C.3: Main modules of the software architecture used for simulation tests.

While the robot is mainly used to collect data and execute motion commands, computation (scan matching, mapping, motion planning) and graphics display may be performed by other computers such as a PC or a laptop. We mainly use a PC equipped with a Pentium III 900 MHZ CPU and 256 M memory to carry out the mapping and reactive motion planning tasks. For hierarchical path planning, which involves heavier computation load due to graph search and additional mapping and graphics display, we use a more powerful PC which is equipped with a Pentium IV 2.4 GHZ CPU and 1 G memory.

Appendix D

Author's Publications

Journal Papers:

[**1**] S. S. Ge, **X. C. Lai**, and A. A. Mamun, "Boundary following and globally convergent path planning using instant goals", *IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics*, vol. 35, no. 2, pp. 240-254, April 2005.

[**2**] S. S. Ge, **X. C. Lai**, and A. A. Mamun, "Sensor-based path planning for nonholonomic mobile robots subject to dynamic constraints", *Robotics and Autonomous Systems*, vol. 55, no. 7, pp. 513-526, July 2007.

[**3**] **X. C. Lai**, S. S. Ge, and A. A. Mamun, "Hierarchical incremental path planning and situation-dependent optimized dynamic motion planning considering accelerations", *IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics*, vol. 37, no. 6, December 2007.

Conference Papers:

[**1**]**X. C. Lai**, C. Y. Kong, S. S. Ge, and A. A. Mamun, "Map building for autonomous mobile robots by fusing laser and sonar data", *Proceedings of 2005 IEEE International Conference on Mechatronics and Automation*, Niagara Falls, Canada, pp. 993-998, 2005. (Best Student Conference Paper Award Finalist)

[**2**] **X. C. Lai**, S. S. Ge, P. T. Ong, and A. A. Mamun, "Incremental path planning using partial map information for mobile robots", *Proceedings of the 9th International Conference on Control, Automation, Robotics and Vision, (ICARCV 2006)*, Singapore, pp. 133-138, 2006.

[**3**] Z. P. Wang, S. S. Ge, T. H. Lee, and **X. C. Lai**, "Adaptive smart neural network tracking control of wheeled mobile robots", *Proceedings of the 9th International Conference on Control, Automation, Robotics and Vision, (ICARCV 2006)*, Singapore, pp. 983-988, 2006.

[**4**] **X. C. Lai**, A. A. Mamun, and S. S. Ge, "Polar polynomial curve for smooth, collision-free path generation between two arbitrary configurations for nonholonomic robots", *Proceedings of the 22nd IEEE International Symposium on Intelligent Control (ISIC2007)*, Singapore, 2007.